VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizace interiéru budov

Architectural Interior Visualization

2017 Filip Moták

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Filip Moták**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Vizualizace interiérů budov
Architectural Interior Visualization

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je navrhnout a naimplementovat aplikaci pro fotorealistickou interaktivní vizualizaci virtuálního nábytku a doplňků interiérů pomocí prostředků rozšířené reality. Předpokládá se implementace demonstrativní aplikace s využitím Unreal Engine (UE) nebo Unity 3D s možností umísťování uživatelských předmětů do reálného interiéru.

1. Seznamte se s technikami vizualizace interiérů v prostředí herních enginů.
2. Vytvořte sadu materiálů pro jejich následnou aplikaci do modelu interiéru.
3. Nastavte vhodným způsobem osvětlení objektů s ohledem na jejich umístění do reálné scény.
5. Vytvořte aplikaci využívající zvolený herní engine a knihovnu pro podporu rozšířené reality umožňující vložení uživatelského objektu do reálné scény pomocí markerů nebo obdobné techniky zajišťující korespondenci mezi umělým a reálným obrazem.
6. Funkcionalitu výsledné aplikace předveďte na vhodné demonstraci zahrnující minimálně promítnutí vybraného objektu do určeného místa v reálné scéně.
7. Použité postupy pečlivě zdokumentujte v textové části práce a výsledky zhodnoťte.

Seznam doporučené odborné literatury:

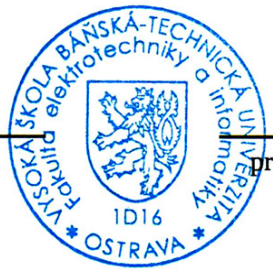
[1] Tavakkoli, A. Game Development and Simulation with Unreal Technology. AK Peters, ISBN 978-1498706247, 741 p., 2015.
[2] Sewell, B. Blueprints Visual Scripting for Unreal Engine. Packt Publishing, ISBN 978-1785286018, 188 p., 2015.
[3] Schmalstieg, D., Hollerer, T. Augmented Reality: Principles and Practice. Addison-Wesley Professional. ISBN 978-0321883575, 528 p., 2016.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval všem, kteří mi pomohli k úspěšnému dokončení této práce, zejména panu Ing. Tomáši Fabiánovi, Ph.D. za cenné rady a rodině a přátelům za podporu.

Abstrakt

Tato diplomová práce se zabývá průzkumem využití virtuální a rozšířené reality v oboru interiérového designu. Popsány jsou také herní enginy jako Unreal Engine a Unity, které se pro tyto účely používají. V práci jsou dále zmíněny využití těchto technologií v ostatních oborech. Implementační část se zabývá vývojem programu pro vizualizaci interiéru s využitím rozšířené reality s použitím knihovny ARToolKit v enginu Unity.

Klíčová slova: vizualizace, interiér, vizualizace interiéru, rozšířená realita, virtuální realita

Abstract

This thesis focuses on current uses of virtual and augmented reality in interior design. Currently popular game engines used for this purpose as Unity and Unreal Engine are also described. The thesis further focuses on current uses of said technologies in other fields. Implementation part focuses on interior visualization augmented reality application development using Unity with ARToolKit library.

Key Words: visualization, interior, interior visualization, augmented reality, virtual reality

Obsah

Seznam použitých zkratek a symbolů

AR	–	Augmented Reality
VR	–	Virtual Reality
UE4	–	Unreal Engine 4
UWP	–	Universal Windows Platform

Seznam obrázků

Seznam tabulek

1 Úvod

Počítačová vizualizace je dnes široce používanou technikou v oborech exteriérového a interiérového designu. Umožnění zobrazit výsledný produkt zákazníkům před samotnou realizací je velmi ceněnou službou, kterou najdeme v repertoáru téměř každého interiérových studia. V této práci se zaměříme na pokročilejší praktiky, které statickou vizualizaci rozšiřují o interaktivní prvky se snahou zvýšit uživatelovu imersi a vnést tak do interiérové vizualizace nový rozměr. Toho se budeme snažit docílit dvěma technikami: virtuální (VR) a rozšířenou (AR) realitou. V první části práce se zaměříme na realitu virtuální, její použití v jiných oborech a následnou aplikaci ve vizualizaci interiérů s použitím herního enginu Unreal Engine. V druhé části si pak popíšeme využití rozšířené reality v různých oborech a pro potřeby vizualizace interiéru s následným popisem implementace demonstrační aplikace v enginu Unity.

Snahou této práce bude porovnat tyto dvě metody, srovnat jejich klady, zápory a popsat různé praktiky, které se v daných technologiích používají. Programovým výsledkem bude aplikace využívající rozšířené reality pro vizualizaci interiérových prvků.

1.1 Rozšířená vs. virtuální realita

Rozšířená realita spolu s realitou virtuální v posledních letech zaznamenávají velký vzrůst v popularitě. Nicméně jsou tyto dvě technologie často nesprávně zaměňovány. Řekněme si tedy prvně, co vlastně virtuální a rozšířená realita jsou.

Dle slovníku Merriam-Webster je virtuální realita definována následovně [1]:

> Virtuální realita - umělé prostředí, jež je zažité skrz smyslové podněty (jako zrak a sluch), vytvořené počítačem, který rozhoduje o tom, co se v tomto prostředí stane.

Pokud mluvíme o virtuální realitě, uvažujeme tedy počítačem vytvořený svět, který zaznamenáváme alespoň zrakem a sluchem, většinou s použitím k tomu určenému hardwaru (náhlavní souprava, displaye). Za vzrůst v popularizaci této technologie se zasloužil hlavně pokrok v možnostech počítačové grafiky, kde je možné vykreslit obraz téměř nerozeznatelný od reality, a to v rozumném čase.

Jaký je tedy rozdíl mezi virtuální a rozšířenou realitou? Merriam-Webster definuje AR následovně [2]:

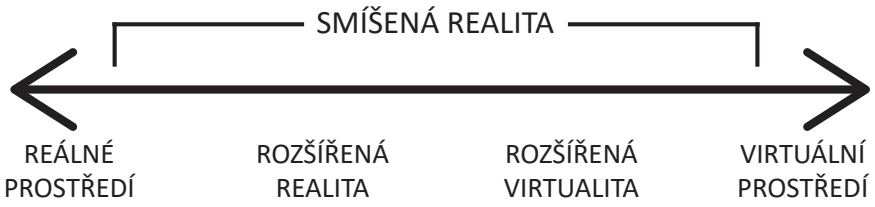
> Rozšířená realita - obohacená verze reality s použitím technologie umožňující překrýt digitální informace přes obraz, zachycený pomocí nějakého zařízení (kamera, telefon či tablet, apod.).

Z této definice by se mohlo zdát, že by zde mohly spadat také filmy se speciálními efekty, jako například série Harry Potter a Star Wars nebo 2D overlaye, a proto bychom se přiklonili k definici dle Azumy [3], která říká, že systémy využívající rozšířenou realitu mají tyto tři základní rysy:

- kombinují realitu a virtualitu,
- umožňují interakci v reálném čase,
- jsou registrované ve 3D.

Takto odpadá podmínka omezení na jakékoliv technologie a klade se důraz na interaktivitu, která je hlavním faktorem AR aplikací.

Představíme-li si realitu jako osu (obr. 1), kdy na jedné straně leží reálné prostředí a na druhé realita plně virtuální, AR bude ležet kdesi uprostřed. Poměr virtuálních prvků vůči těm reálným pak tuto technologie posouvá na jednu či druhou stranu. Toto znázornění je nazýváno *virtuální kontinuum* a bylo prvně představeno Paulem Milgramem a kolektivem [4].



Obrázek 1: Virtuální kontinuum - pomyslná osa mezi reálným prostředím a rozšířenou realitou.

Kromě virtuální a rozšířené reality se definuje také tzv. rozšířená virtualita. Ve výsledku se jedná o jakýsi protiklad reality rozšířené, kdy jsou do virtuálního prostředí zobrazovány prvky reálného světa. Reálnými prvky mohou být textury nebo i video. Tuto techniku využívá například systém „cAR/PE!“, který do počítačem vytvořeného prostředí umisťuje webovou kamerou zachycené video všech účastníků v reálném čase. V moderních aplikacích se experimentuje s využitím náhlavních souprav pro virtuální realitu. Příkladem může být aplikace, která do virtuálního prostředí dokáže vložit uživatelovy ruce a ty následně využít pro ovládání interaktivních prvků. Pro tento účel je na headset připevněna hloubková kamera.

2 Virtuální realita

Jak již bylo zmíněno, virtuální realita se v dnešní době stává více a více populárním médiem pro zábavu, výuku a vizualizaci, především díky stále se zdokonalujícímu hardware, umožnující uživateli zobrazit virtuální prostředí, které je téměř nerozeznatelné od reality díky vysoce kvalitním displayům, pohybovým detektorům a software umožnujících tuto vizualizaci v reálném čase. V době psaní této práce jsou na trhu virtuální reality k dispozici tři náhlavní soupravy přímo určené pro VR: HTC Vive, Oculus Rift a Playstation VR. Jedná se o headset s displayem pro každé oko, který je kabelem přímo spojen s počítačem, jež zpracovává veškerý vstup a výstup. Za vstup jsou zde považujeme uživatelovy pohyby hlavou, případně dalšího příslušenství, jako ovladače, gamepady, apod. Oproti tomu je na trhu dostupný velký počet tzv. mobilních headsetů, do kterých je možné vložit mobilní telefon, který se stará jak o vstup a výstup přímo, bez nutnosti být připojen k externímu hardware. Toto řešení ovšem disponuje menší výkonností a horším detailem obrazu [5]. V České republice se v době psaní této práce ceny mobilních náhlavních souprav pohybují kolem 1 500 Kč, dedikované soupravy pak kolem 20 000 Kč.

Ruku v ruce se zvyšující se kvalitou hardware jde také programové vybavení dnešních počítačů. Unreal Engine 4 a Unity jsou dva herní enginy, které ze základu VR podporují. Hlavním cílem těchto enginů je dát vývojářům nástroj pro tvorbu aplikací virtuální reality, přičemž mohou využít předchozích znalostí z 3D herního vývoje.

Nevýhody, které s sebou virtuální realita nese, přichází v podobě nevolnosti, kterou uživatelé mohou zažít při užívání VR headsetů. Takový stav je nazýván motion sickness, cybersickness nebo virtual reality sickness. Existuje několik teorií, nejakceptovanější z nich je Sensory Conflict Theory, která říká, že rozdíly mezi smysly podávající informace o orientaci těla vytvoří percepční konflikt a mozek neví, jak jej zvládat [6]. Vývojáři aplikací pro VR se snaží riziko této nevolnosti minimalizovat.

2.1 Existující použití virtuální reality

Virtuální realita nachází využití v mnoha oborech. Zřejmým kandidátem je zábavní průmysl, ale svůj účel si VR najde i ve výuce, tréningu a simulaci. V následujícím textu popíšeme využití v medicíně, armádě a herním průmyslu.

2.1.1 Medicína

Medicína je jedna z největších oblastí využívajících technologií rozšířené a virtuální reality. Virtuálních prostředí se zde hlavně využívá pro simulace chirurgických zákroků, tréning nových technik, ale také při terapiích pro léčení fóbií. Není tedy třeba žádných invazivních zákroků a vše lze provést v bezpečném prostředí, bez jakéhokoliv nebezpečí škody na zdraví pacienta [7].

2.1.2 Armáda

Zde virtuální realita našla využití zejména ve výuce nových rekrutů. Použita je hlavně v zákrocích, které je normálně nebezpečné zkoušet poprvé v reálném prostředí, jako parašutistické seskoky, letecké simulátory, apod. Cílem tohoto tréningu je rekruty naučit jak reagovat v reálných situacích [8].

Dalším příkladem využití může být systém AVCATT-A (Aviation Combined Arms Tactical Trainer-Aviation), simulační software americké armády pro tréning bojových situací v helikoptérách typu Apache AH-64A, AH-64D Longbow a dalších. Nejedná se ovšem o letecký simulátor, systém je spíše využíván schopnými vojáky ke zlepšení svých dovedností. Využívá se zde náhlavních souprav [9].

2.1.3 Zábava

Největší uplatnění virtuální realita nalézá samozřejmě v počítačových a konzolových hrách. Momentálně se většina takových her zaměřuje na jeden typ virtuálního headsetu. Hry pro Occulus Rift a PlayStation VR jsou vyvíjeny s předpokladem, že hráč bude pro kontrolu hry používat gamepad, kdežto HTC Vive používá vlastní ovladače pro každou ruku. Pozice těchto ovladačů je registrována v prostoru a tudíž jejich pohyb dopomáhá k vytvoření iluze reálné interakce s virtuálním obsahem.

Příkladem první metody je hra Resident Evil 7, hororová hra publikovaná firmou Capcom. Hra je vytvořena pro Playstation 4 s podporou Playstation VR headsetu. Hráč se může rozhlížet natáčením hlavy, větší pohyby jako otáčení a ovládání hlavního hrdiny ovšem řeší pouze gamepad.

2.2 Využití ve vizualizaci interiérů

Využití virtuální reality v oboru vizualizace architektury se přímo nabízí. Možnost nechat klienta projít si místnost, byt či stavbu předtím, než budou ve skutečnosti postaveny, je velmi zajímavým prospektem jak pro klienty, tak pro architekty, kteří na daném projektu pracují a kteří tak mohou získat cennou zpětnou vazbu pro vytvoření uspokojivého výsledku.

Chceme-li využít virtuální reality pro vizualizaci interiérů, respektive vizualizaci architektury jako takové, máme dvě možnosti, jak dosáhnout kýženého výsledku. Můžeme sáhnout po komerčních systémech nebo využít herních enginů jak bylo popsáno výše. Vezměme si jako příklad systém Unreal Engine 4. UE 4 je nástroj z dílen firmy Epic Games, zaměřující se nejen na tvorbu videoher, ale také 3D filmů a vizualizací. Můžeme položit otázku, proč používat herní engine pro takové účely, jako zobrazení architektury, když lze na trhu nalézt nástroje, které jsou k tomu účelu přímo určené. Jednou z odpovědí by mohla být cena. Ceny komerčních nástrojů pro vizualizaci architektur se mohou vyšplhat na vysoké částky, například za software Lumion můžeme zaplatit až 1 499 € v limitované verzi, 2 999 € pak pro verzi plnou [10]. Unreal Engine 4 je poskytován zcela zdarma pro potřeby vizualizace [11].

Další možnou odpovědí na danou otázku by mohly být zobrazovací možnosti Unreal Enginu. Jakožto herní engine uzpůsobený pro zobrazování v reálném čase, UE poskytuje uživateli nebo zákazníku možnost projít a prohlédnout si scénu zevnitř, zatímco u ostatních softwarů by se musel spokojit s obrázky či animacemi, maximálně 360° panoramaty. [10].

Kde Unreal Engine lehce zaostává za konkurencí komerčních softwarů pro vizualizaci, je kvalita tohoto zobrazení. Software jako Lumion jsou přímo uzpůsobeny pro zobrazování interiérů a exteriérů, jsou tedy vybaveny všemi potřebnými nástroji pro snadné vytvoření realistického výsledku světel, stínů a materiálů. Jelikož je často výsledkem statický obraz, mohou být pro zobrazení použity metody, které se v realtimové grafice realizují obtížně (např. raytracing, pathtracing).

Obrázek 2: Příklady realistických vizualizací interiérů vytvořených v herním enginu Unreal Engine. (zdroj: Unreal Engine Marketplace - Xoio Berlin Flat, Realistic Rendering).

3 Vizualizace interiérů v Unreal Engine 4

Této práci předcházel semestrální projekt, jehož cílem bylo prozkoumat možnosti herních enginů pro účely vizualizace interiéru. Úkolem bylo vytvořit demonstrační aplikaci a porovnat výhody a nevýhody dané techniky. Zvoleným enginem byl výše zmíněný Unreal Engine 4. V následujícím textu se budeme věnovat popisu principů a praktik, které byly v práci využity.

3.1 Příprava vstupních souborů

Před samotnou vizualizací musíme předem připravit veškeré modely a textury. Plánujeme-li uživatelům umožnit virtuální prohlídku dané scény, je třeba dbát na dostatečnou kvalitu těchto prvků, protože právě tyto (společně s osvětlením) hlavně ovlivňují jejich pohroužení a celkový dojem. Všechny takové modely, textury, materiály, světla a kamery v Unreal Engine 4 vystupují jako tzv. *Asset* [12].

Unreal Engine u modelů požaduje druhý kanál pro UV jako kanál pro lightmapping, který se stará o správný výpočet osvětlení. Je tedy nutné model pro import připravit tak, aby obsahoval oba tyto kanály se správnými mapami. K tomuto účelu byl použit Autodesk 3ds Max 2016 Student Edition.

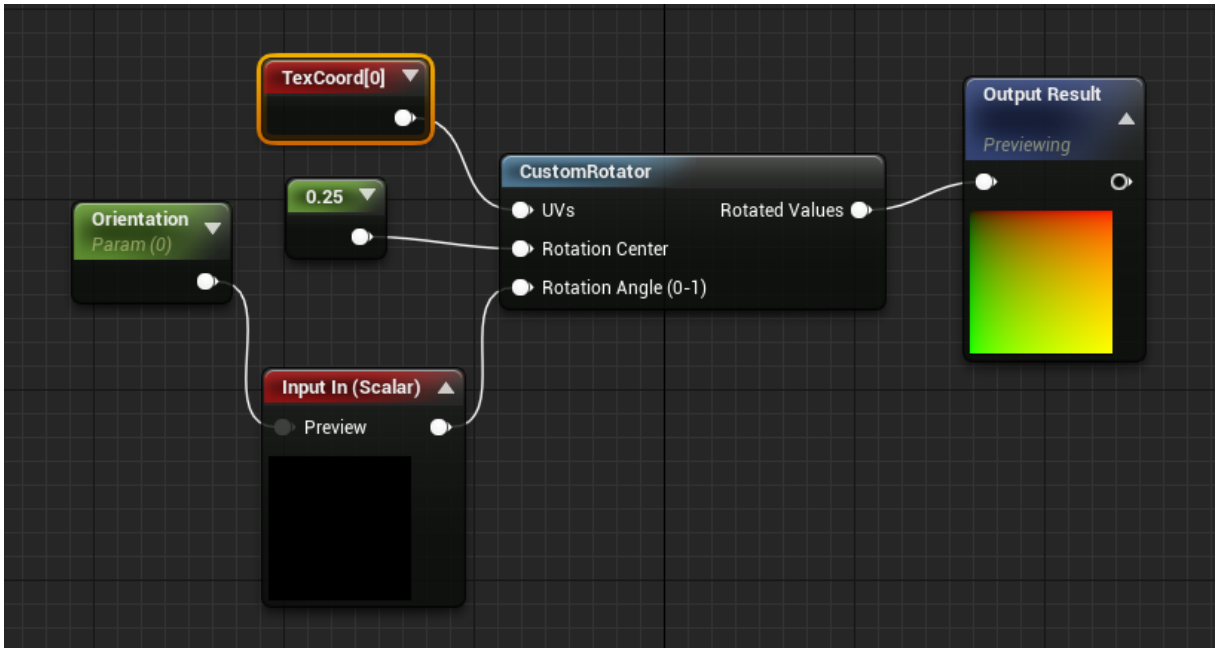
3.2 Materiály a meteriálové funkce

Scéna vyžaduje množství materiálů pro různé účely, v našem případě například materiály dřeva pro dveře, podlahy a lišty, materiál omítky pro pokrytí zdí a ostatní, jako například sklo a kov pro jiné prvky. K tomuto účelu poskytuje UE4 velmi výkonný *Material editor* - grafické rozhraní, které nám umožňuje snadno a rychle vytvořit vysoce parametrizovatelné shadery pro aplikaci na modely [13, 14].

Často používané části shaderů můžeme ukládat do tzv. Material Functions: parametrizovatelné bloky, které lze jednoduše a opakovaně aplikovat. Změny v materiálové funkci jsou následně propagovány do všech bloků, kde je tato funkce použita. Příklad lze vidět na obr. 3.

3.3 Osvětlení

Osvětlení scény hraje velkou roli při snaze o realistickou vizualizaci a může se projevit jako největší překážka k docílení kýženého výsledku. Unreal Engine poskytuje nástroje pro přímé i globální (nepřímé) osvětlení. Editor UE 4 nám dává k dispozici tři základní typy světelných zdrojů: *Directional Light*, *Point Light* a *Spot Light* [15]. Každý z těchto typů má své využití. Scéna obvykle obsahuje jednu instanci Directional Light, která simuluje vnější zdroj světla (typicky Slunce nebo Měsíc), tedy zdroj, který je pro jednoduchost nekonečně vzdálený. Spot Light a Point Light jsou nejčastěji používané jako umělé zdroje světla. Point Light vyzařuje světlo do všech stran, hodí se tedy pro lustry a žárovky. Oproti tomu Spot Light světlo vyzařuje jako kužel, použití najde pro reflektory nebo bodovky. Globální osvětlení realizuje iluminaci scény



Obrázek 3: Materiálová funkce implementovaná v materiálovém editoru Unreal Enginu. Veškeré shadery jsou zde řešeny stejně, prostřednictvím funkčních uzlů a jejich vzájemným propojováním.

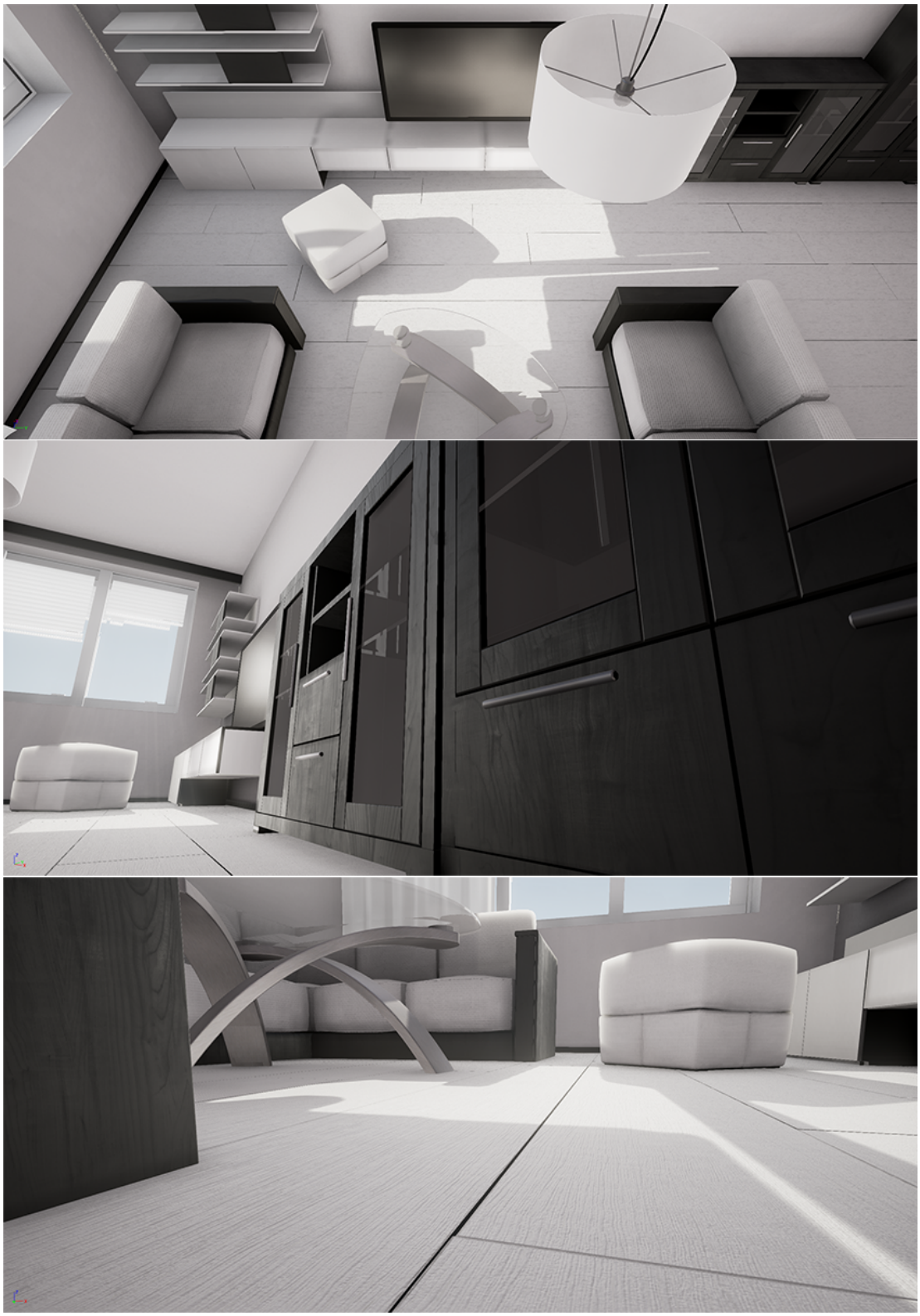
pomocí výpočtů odraženého světla ze zdrojů od povrchů interiéru. Pro jeho realizaci a kontrolu můžeme použít tzv. *lightmass*, neboli statický systém pro globální osvětlení [16].

3.4 Interaktivní prvky

Pro prohloubení uživatelova zážitku můžeme scénu obohatit o interaktivní prvky jako vypínače, zásuvky, otevíratelné skříňky, apod. a nechat jej tak si scénu „osahat“. Kombinací všech těchto assetů získáme konečnou virtuální scénu.

3.5 Výsledná virtuální scéna

Na následujících obrázcích (obr. 4) můžeme vidět příklad výsledku scény připravené pro virtuální prohlídku, která byla vytvořena v rámci semestrálního projektu. Ve scéně jsou využity parametrizované procedurální generátory pro podlahu a závěsy. Interaktivním prvkem jsou zde zatažitelné a otevíratelné žaluzie.



Obrázek 4: Výsledná scéna vytvořená v Unreal Engine 4. Na obrázcích lze vidět výsledek výpočtu osvětlení. Podlaha na třetím obrázku je generována tak, aby jednotlivé desky mezi sebou obsahovaly různé mezery, což umocňuje realistický dojem.

4 Rozšířená realita

V sekci 1.1 jsme definovali AR. Nyní si ji popíšeme podrobně, seznámíme se se základními principy a představíme si možnosti užití. Dále si řekneme jaké metody se používají pro sledování kamery v prostoru a jejich princip.

Rozšířená realita není omezena pouze na zobrazovací systémy. Zvukové, haptické, či jiné technologie, které kombinují dvě reality, splňují definici i když jsou možná složitěji realizovatelné [17]. V této práci se ovšem zaměříme pouze na vizuální systémy.

4.1 Nástin problematiky

Jako nejjednodušší příklad si můžeme vzít umístění obyčejné kostky do prostoru. Pro usnadnění si představme případ, kdy je prostor snímán pomocí jedné kamery a my na něj nahlížíme zpoza obrazovky. Naším cílem je vytvoření iluze existence tohoto virtuálního objektu (v našem případě kostky). Můžeme si tedy definovat několik požadavků, které by takto vizualizovaný objekt měl splňovat:

- realistické umístění v prostoru,
- možnost nahlédnout na objekt z různých stran,
- viditelnost částí objektu, pokud celý nespadá do záběru,
- stabilita při pohybu kamery,
- realistické osvětlení objektu,
- realistická textura, apod.

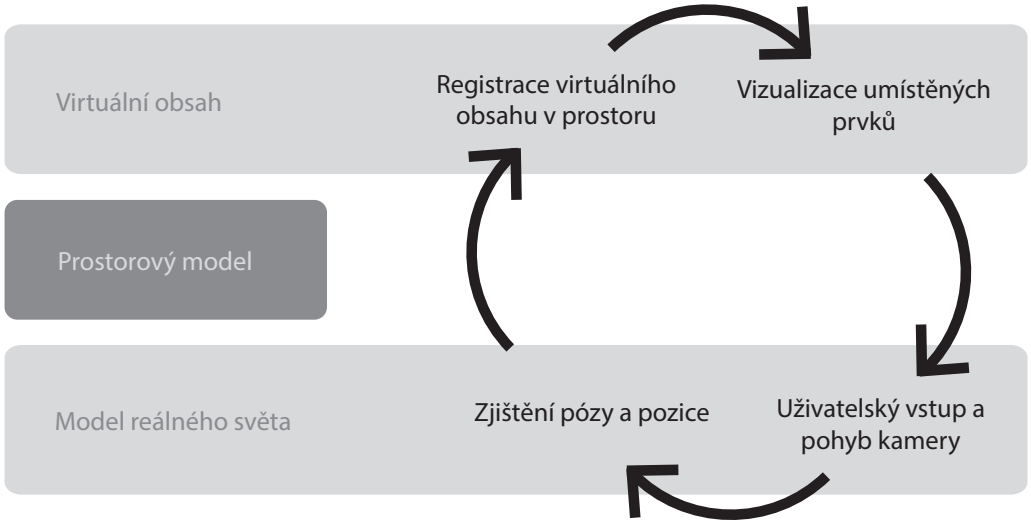
Takovýchto podmínek můžeme vymyslet velké množství. A právě realizací těchto vlastností se při tvorbě AR aplikací zabýváme. V dalších sekcích projdeme teorii jednotlivých problematik společně s příklady.

4.2 Určení pozice kamery

Základní problematikou vizuální rozšířené reality je určení pozice kamery vzhledem k virtuální scéně. V našem příkladě s kostkou chceme, aby se při pohybu kamery reálně přizpůsobil také zobrazovaný objekt, tj. při přiblížení se kostka zvětší, při natočení se posune a rotuje apod. Musíme tedy vytvořit jakýsi systém, který nám umožňuje tyto pohyby z obrazových dat rozeznávat a fakticky umožnit programu orientovat se v prostoru bez použití hloubkových, či jiných technologií.

Postup kroků k docílení efektu rozšířené reality je stejný ve všech systémech. Jedná se smyčku zpětné vazby mezi uživatelem a počítačem (obr. 5), kdy uživatel sleduje veškeré dění s pomocí

kamery či jiného zařízení a ovládá jeho pohled. Program v odpovědi na tento vstup sleduje jeho pozici vzhledem k reálnému světu a uživateli zpětně zobrazí scénu obohacenou o virtuální prvky. Máme tedy smyčku obsahující čtyři základní kroky [17].



Obrázek 5: Zpětnovazební smyčka zpracování rozšířené reality.

Chceme-li zjistit pozici kamery v prostoru, musíme z obrazového vstupu získat zájmové body, které nám při pohybu kamery umožní výpočet její polohy a úhlu pohledu. Máme celkem dvě možnosti jak reálný prostor mapovat, a to takové, které využívají externích pomocných markerů či jiných objektů a bezmarkerové systémy, které využívají převážně čistý obrazový vstup.

4.2.1 Markerové systémy

Značky, nebo také markery, jsou kontrastní předem známé obrazce, které jsou v obraze lehce vyhledatelné, identifikovatelné, ideálně unikátní a zjednodušují tak vyhledávání jinak složitých zájmových bodů [18]. Většinou se jedná o čtvercové symboly, ale najdeme i kruhové či jiné varianty. Příklady markerů jsou na obrázku 6.

Algoritmy pro vyhledávání markerů se mohou v různých knihovnách a řešeních lišit, princip většinou ale zůstává stejný. Příklad, který zde uvádíme je použit v knihovně ARToolKit. Pro vyhledání markeru je obraz převeden do binární formy, většinou některou z metod pro prahování, ať už s pevným či adaptivním prahem. V takovém obraze jsou následně detekovány hrany odpovídající obálce (okraji) daného markeru. Poté je proveden matematický výpočet relativní pozice a orientace ke kameře. Posledním krokem je identifikace markeru, kdy program získá obrazec uvnitř jeho okraje a následně jej porovná s předem registrovaným seznamem vzorů. Na takto identifikovaný a popsaný marker je již možno renderovat virtuální obsah [19].

Obrázek 6: Kontrastní markery - vlevo můžeme vidět klasický černobílý marker, uprostřed pak marker barevný s identifikačním kódem po obvodu. Vpravo můžeme vidět kruhový design.

Výhodou markerových systému je velmi rychlá detekce a jednoduchá identifikace s malým procentem falešných pozitiv [18]. Tato řešení dokáží pracovat s velkým počtem různých obrazců bez většího vlivu na výkon aplikace. Nevýhodou je „špinavý obraz“ - nutnost použití markerů může znehodnotit uživatelovu zkušenost s AR.

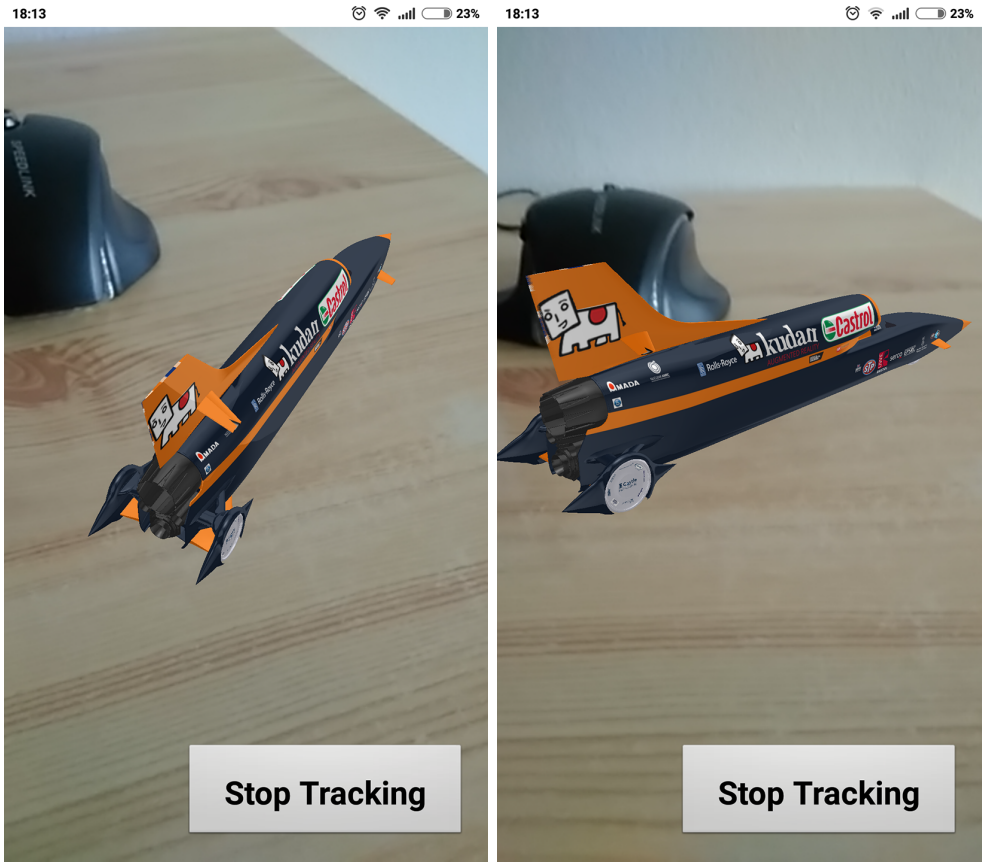
4.2.2 Bezmarkerové systémy

Nechceme-li si scénu „špinit“ nepřirozenými markery, můžeme sáhnout po metodách, které využívají samotný obraz (případně akcelerometru a kompasu zařízení) pro určení pozice kamery. Možnou realizací bezmarkerového mapování prostoru je využití tzv. natural features, zájmových bodů okolí. V tomto případě hledáme korespondenci mezi 2D body v obraze, které jsme získali z kamery a známými 3D body v prostoru.

Jednou z možností určení pózy je přímá detekce, kdy pozici určujeme v každém rámci obrazu znovu bez toho, aniž by se spoléhalo na jakékoliv rámce předchozí. Pro známé zájmové body vytvoříme tzv. deskriptory, popisné struktury, které umožňují rychlé a spolehlivé porovnání. Tyto deskriptory také vytvoříme pro zájmové body v obrazu kamery a následně porovnáme se známým modelem. Pro získání zájmových bodů můžeme použít metody jako Harrisův detektor hran, rozdíl Gaussiánů, FAST, apod. Pro vytvoření deskriptorů je vhodná např. metoda SIFT nebo SURF [17].

V okruhu bezmarkerových systému zaujímá velice zajímavou pozici systém SLAM. SLAM je zkratka pro Simultaneous Location and Mapping, systém jež se snaží zároveň mapovat prostředí společně s výpočtem polohy v daném prostoru. Tohoto můžeme docílit použitím obrazu samotného, nicméně v praxi systému napomáháme dalšími senzory, například přídavnými kamerami, hloubkovými čidly, gyroskopy, apod. Výhodou tohoto principu je možnost pracovat v neznámém prostředí, bez nutnosti jakékoliv předchozí znalosti zájmových bodů. Nevýhodou je vysoká výpočetní náročnost [20]. Příklad aplikace využívající SLAM lze vidět na obrázku 7.

Uvažujme implementaci SLAMu s jednou kamerou. Kamera je umístěna do prostoru, zaměřena na známý objekt (například kus papíru). Systém si tak dokáže určit metriky pro další operace. Následně s kamerou pohneme a tím začneme mapovat neznámé prostředí a lokalizovat



Obrázek 7: Příklad bezmarkerového určení pozice kamery pomocí SLAMu (zdroj: Kudan Simple Samples).

pozici kamery. To se provádí detekcí zájmových bodů (např. hran, rohů nebo čar) a jejich pohybu při rotaci kamery. Zaměříme-li zmapované body, pak jsme schopni lokalizovat kameru. Základní vlastnosti tohoto algoritmu by měla být opakovatelná lokalizace, tzn. vlastnost systému určit pozici kamery v 10 minutách běhu se stejnou přesností, jako v prvních 10 sekundách. Nedodržením této vlastnosti dochází k tzv. driftu - skluzu mapovaných dat. Tomu se vyhneme vhodnou volbou dobře identifikovatelných, charakteristických bodů [21].

4.3 Viditelnost objektů při zakrytí

Za další požadovanou funkčnost bychom mohli považovat zneviditelnění modelu nebo jeho části, dojde-li k překryvu s reálným objektem (např. zeď). Většina AR systému pouze zobrazuje virtuální prvky přes obraz a tímto problémem se nezabývá. Jedná se o komplexní problém, jelikož o prostředí, které chceme rozšiřovat, moc nevíme [22].

Pokud vizualizujeme statické prostředí, můžeme si předem připravit jeho model nebo hloubkovou mapu a využít těchto znalostí pro výpočet okluze. Toto ovšem není vždy možné, někdy i nereálné v případě složitých scén. Hledáme proto jiné metody, které by tuto funkčnost umožnily

bez nutnosti předchozí znalosti. Jiná metoda řešení využívá technik pro získání hloubkové mapy v reálném čase, jedná se tedy o dynamické řešení okluzí. Takovou mapu lze získat mnoha způsoby, například stereoskopickými metodami [23] či hloubkovými infračervenými čidly. Výsledná hloubková data využijeme ke zjištění pozice virtuálního objektu v prostoru a k výpočtu okluzí.

Představme si jeden způsob řešení. V jednoduchých systémech je obraz videa promítán na rovinu, přes niž je registrován virtuální obsah. Pro realizaci okluzí [24] využívá RGB dat ze senzoru Kinect, kdy každý pixel je posunut vpřed dle hodnoty získané z hloubkové mapy a následně zobrazujeme objekty v tomto prostoru. Musíme zde také řešit problém se zobrazením, jelikož virtuální obsah využívá perspektivní projekci, zatímco obraz kamery musí být renderován v pravoúhlém promítání tak, aby byly pixely zarovnány s rovinou XY a nešel vidět rozdíl hloubek.

Okluze hrají velkou roli při realistické vizualizaci objektů a dále prohlubují uživatelův zážitek z AR, nicméně pro správnou funkčnost vyžadují pomocné senzory.

4.4 Existující použití rozšířené reality

Využití AR našlo převážně ve výuce a instruktáži, kdy není nutné modelovat celou scénu, jak tomu bylo v případě virtuální reality, ale překryjeme existující scénu pomocnými prvky, jako jsou šipky, ukazatele a notace. Svůj účel rozšířená realita samozřejmě našla také v zábavním průmyslu.

4.4.1 Medicína

Jedním z využití rozšířené reality je v oboru medicíny. AR můžeme využít pro účely tréninku a vizualizací postupů při operacích či jiných zákrocích. Nabízí se možnost využít 3D data pacienta, získaná např. magnetickou rezonancí či ultrazvukem a následně je vizualizovat v reálném prostoru. Chirurg by tak byl schopen vidět vnitřní orgány, kosti a tkáň v přesném umístění, bez nutnosti využití invazivních metod.

V roce 1994 na Univerzitě v Severní Karolíně, Chapel Hill, prezentovali aplikaci rozšířené reality, která umožnila doktoru sledovat plod přímo v těhotné pacientce [3]. Tento byl snímán pomocí ultrazvuku a vizualizován na náhlavní soupravě. Dalším zkoumaným využitím je aplikace pro navádění jehly provádějící biopsii tkáně.

4.4.2 Údržby a opravy

Široké možnosti využití nalézá AR v oblasti opravy a údržby automobilů, letadel či jiných vozidel, případně montáži složitých strojů. Technikovi v této situaci můžeme předat instrukce srozumitelnou formou v podobě 3D obrazců, šipek či jiných pomocných elementů a notací. Tyto objekty mohou být animovány, prohlubujíce tak názornost daného procesu.

Jednou z prvních aplikací tohoto typu byl systém KARMA (Knowledge-based Augmented Reality Maintenance Assistant), představen v roce 1993 panem Feinerem a kolektivem. Tento

systém využíval náhlavních souprav pro zobrazení akcí, které je nutné provést pro údržbu laserové tiskárny [17].

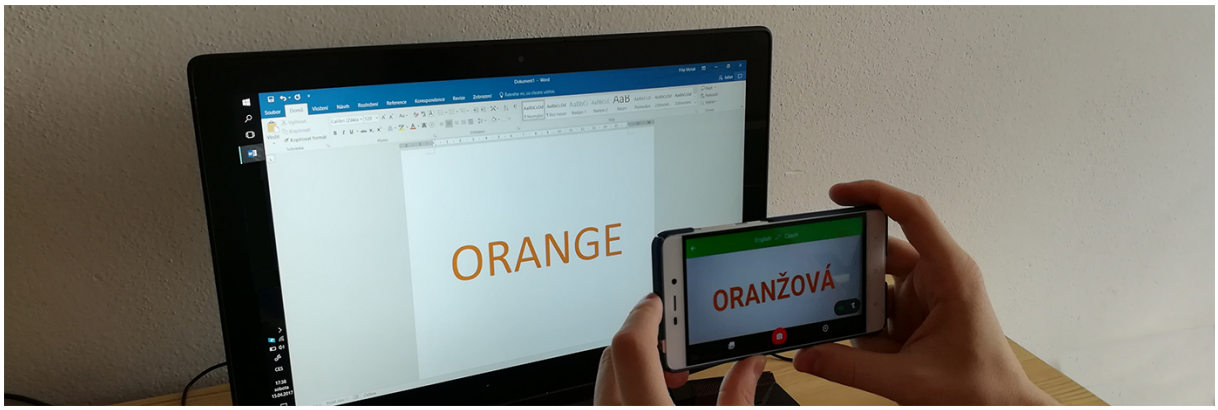
4.4.3 Informace a vzdělávání

Ve školství AR nalézá uplatnění při výuce složitých trojrozměrných modelů, grafů či jiných objektů nebo technik, které nelze snadno zobrazit na standardní média. Lze využít kolaborativních AR aplikací, jako Studierstube [25], systém vyvinutý Schmalstiegem a kolektivem v roce 1996. Tato aplikace umožňovala vizualizovat virtuální objekty více osobám ve společném prostoru. Systém byl v roce 2003 úspěšně vyzkoušen při výuce geometrie pro středoškolské studenty.

Jako další využití se nabízí aplikace pro navigaci a seznámení se s určitým prostorem. Hlavní myšlenkou je vytvoření systému, který z okolí dokáže rozeznat pozici uživatele a ukázat mu cestu k požadovanému místu, únikovému východu, letištnímu terminálu, atd. Do této kategorie také spadají klasické automobilové navigace či parkovací asistenti.

Aplikace jako Yelp Monocle překrývají obraz kamery různými informacemi, v případě této aplikace hodnocením restaurací, hotelů apod. Tyto informace jsou staženy z internetu a dle GPS souřadnic uživatele a pozice telefonu jsou umístěny do reálného prostoru.

Posledním příkladem využití rozšířené reality v informacích je možnost aplikace Google Translate (obr. 8), která v reálném čase nahrazuje překládaný text jeho přeloženou variantou, přičemž volí také podobný typ a velikost písma a barvu pozadí jako u originálního nápisu.



Obrázek 8: Překlad v reálném čase s využitím rozšířené reality pomocí Google Translate mobilní aplikace. Povšimněme si snahy zvolit stejný styl a barvu písma překladu.

4.4.4 Zábava

AR samozřejmě vzbudil velký zájem v okruhu mobilních a počítačových her. Možnost vkládat virtuální obsah do reálného světa dala za vznik hrám jako Pokémon Go společnosti Niantic, mobilní aplikaci zasazené do prostoru Japonské herní série (obr. 9). Hra rozšiřuje mapu reálného světa o virtuální prvky jako stadiony, stanice a samotná stvoření, které se uživatel snaží získat.

Obrázek 9: Pokémon Go - mobilní hra společnosti Niantic. Zde můžeme sledovat hned dvě použití rozšířené reality. Na obrázku vlevo je mapa reálného světa rozšířena o virtuální stadion, na obrázku vpravo je pak zobrazen Pokémon v reálném prostředí. (zdroj: Aplikace Pokemon Go)

4.4.5 Vizualizace interiérů

Jestliže se chceme bavit o využití AR ve vizualizaci interiérů, narazíme na hotová řešení, která dokáží do virtuální scény umístit kusy nábytku. Většinou se jedná o různé katalogy, které uživatelům umožňují vyzkoušet si, jak daný nábytek bude vypadat v jejich domově. Jedním z vývojářů těchto aplikací je společnost Augment, která poskytuje nástroj pro vizualizaci a prezentaci daných modelů. Součástí je také program pro jejich správu [26].

Švédský nábytkářský gigant Ikea nezůstává pozadu a od roku 2012 poskytuje ke svému katalogu aplikaci pro rozšířenou realitu, kdy si uživatelé mohou vizualizovat vybrané kusy nábytku na hlavní straně katalogu, jež zde slouží jako marker.

Trochu jiný přístup k využití rozšířené reality v designu zvolila nizozemská společnost AkzoNobel. Ve své aplikaci Dulux Visualizer umožňuje uživateli vybrat z širokého repertoáru barev, které jsou následně virtuálně aplikovány na zdi interiéru. Uživatelé si tak mohou vyzkoušet danou barvu předtím, než si ji koupí. Aplikace se také pyšní funkcí, která uživateli vybere vhodné barevné schéma [28].

5 Knihovny pro rozšířenou realitu

Vývojáři AR aplikací nemusí takové programy stavět od začátku. Existuje řada knihoven, které se touto problematikou zabývají. Mnoho z nich poskytuje SDK pro různé platformy a jazyky, případně plug-iny pro herní enginy. V dalších sekcích zmíníme knihovny ARToolKit, EasyAR, Kudan a Vuforia.

5.1 ARToolkit

Knihovna ARToolkit byla poprve demonstrována v roce 1999 na konferenci SIGGRAPH. V roce 2001 poté vydána jako open source knihovna ve verzi 1.0 společností ARToolworks. ARToolkit se tehdy stal základním kamenem výuky a výzkumu rozšířené reality. Nejnovější verze při psaní tohoto dokumentu je 5.3.2 a knihovna v průběhu let prošla mnohými vylepšeními.

Knihovna ze základu podporuje jazyky C++, Java (Android), Objective C (iOS) a vývoj v herním enginu Unity (C#). ARToolkit je proslulý čistým využitím markerů pro účely sledování pozice kamery.

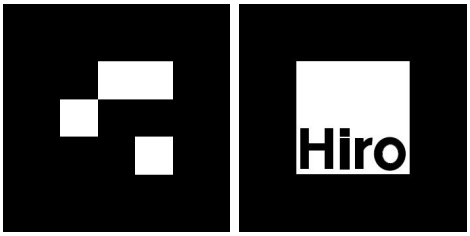
Tradiční čtvercové markery V této knihovně existuje několik markerových typů. První typ využívá předdefinovaných čtvercových markerů, v dokumentaci knihovny pojmenované jako „Traditional Template Square Markers“. Tyto markery přitom musí splňovat určité specifikace, přesněji:

- musí být čtvercové,
- musí mít celistvý okraj (doporučeno je 25% šířky markeru),
- barva popředí a pozadí musí být co nejvíce kontrastní.

Vzor uvnitř okraje může být libovolný a slouží pro jednoznačnou identifikaci. Tento vzor samozřejmě musí být pro každý marker unikátní a asymetrický pro správnou detekci rotace. Také různé barvy markerů mohou napomoct k vylepšení přesnosti detekce. Takto můžeme definovat neomezené množství markerů. Knihovna k tomuto účelu poskytuje sadu pomocných nástrojů. Příklad markeru lze vidět na obrázku 10.

Markery s čárovým kódem Využití libovolných vzorů uvnitř markerů přichází s určitou cenou v podobě doby výpočtu. Tento problém se snaží řešit druhý typ čtvercových markerů, tj. markery s dvourozměrnými čárovými kódy (Barcode) pro vzory uvnitř pole. Tyto jsou součástí knihovny a nemusíme je znovu definovat. ARToolkit je schopen tyto markery najít v konstantním čase, to znamená, že detekce různých markerů zabere stejný čas. Použitím tohoto typu se snižuje šance špatné detekce. Marker lze vidět na obrázku 10.

Oba zmíněné typy markerů trpí vadou, kdy se marker stane nedetekovatelným, jakmile je přerušen či jinak zakryt vnější obvod okraje. Na druhou stranu jsou obě metody velmi stabilní i při rychlých pohybech kamery.



Obrázek 10: Barcode Marker (vlevo) a tradiční čtvercový marker (vpravo).

NFT markery Posledním typem detekce je tzv. Natural Feature Tracking (NFT), metoda, která předem najde v markeru výrazné body (hrany, rohy) a následně je porovnává se vstupním obrazem. Tento typ markerů musíme předem trénovat, výstupem tohoto trénování je právě množina zájmových bodů. Existuje zde znovu pár pravidel pro generování správného NFT markeru:

- musí být čtvercové,
- musí být ve formátu jpeg,
- vstupní obraz musí obsahovat dostatečné množství detailů a hran,
- z obrazu většího rozlišení se dá získat více zájmových bodů.

ARToolkit znovu disponuje sadou programu pro vytváření a trénování NFT markerů. Oproti ostatním metodám je NFT více náročné na výpočet, výhodou ovšem je možnost proměnit libovolný návrh v marker.

5.2 EasyAR

EasyAR, knihovna čínské firmy VisionStar Information Technology, byla prvně vydána v roce 2015. Nejnovější verze v době psaní tohoto dokumentu je 1.3.1, přičemž je ohlášena verze 2.0 přinášející zásadní změny a nové technologie. Knihovna EasyAR se může jevit jako přímý konkurent ARToolkitu. Obě knihovny se zaměřují na čisté použití markerů, EasyAR ovšem volí jednotnou metodu řešení.

Detekce markerů v této knihovně je řešena jedinou metodou, a to hledáním důležitých bodů v předem uložené šabloně (NFT u ARToolkit). Tato metoda umožňuje robustnější detekci markeru i v případě, že je vzor částečně zakryt nebo jinak špatně viditelný. Pro rozpoznání markeru můžeme využít cloudového zpracování. Výhodou této metody je možnost velice jednoduše nahrávat různé markery i za běhu programu, vše co knihovna potřebuje znát je soubor s obrázkem.

Při výběru vhodné knihovny pro řešení této práce byla však detekce markeru shledána pomalou. Mezi jeho odhalením kameře a rozpoznáním byla téměř pětisekundová prodleva. Také stabilita detekce byla poněkud horší něž u knihovny ARToolkit, kdy se 3D model na markeru mihotal nebo nedokázal udržet pevnou velikost. Další nepříjemným zjištěním byla nedostatečná dokumentace s lámanou angličtinou a komunitní fórum pouze v čínštině.

5.3 Kudan

Kudan je britsko-japonská vývojářská společnost zaměřující se na umělou inteligenci a internet of things [29]. Tato knihovna umožňuje jak využití markerů pro určení pozice kamery, tak SLAMu jako bezmarkerové varianty. Kudan neposkytuje SDK pro vývoj nativních počítačových aplikací, nicméně poskytuje nástroje pro nativní vývoj na mobilních zařízeních se systémem Android, iOS a plug-in pro herní engine Unity. Bezplatná verze SDK obsahuje vodoznak.

5.4 Vuforia

Knihovna Vuforia poskytuje SDK pro vývoj na systémech Android, iOS, UWP a herní engine Unity. Je poskytovaná zdarma pro účely vývoje a při nasazení je třeba upgradovat licenci na jednu z placených variant. Knihovna poskytuje mnoho možností jak sledovat objekty v prostoru, přes klasické planární markery, až po 3D objekty. Dalé je možno rozpoznat a sledovat anglická slova ze slovníku cca 10 000 slov [30].

6 Vlastní řešení vizualizace v rozšířené realitě

Dle zadání byla vypracována aplikace pro rozšířenou realitu schopná vizualizovat interiérové prvky do prostoru. Nástin hlavní myšlenky je na obr. 11. Cílem je co nejpřesnější a nejrealističtější zobrazení virtuálních interiérových prvků v prostoru, přičemž se snažíme o pokrytí co největší plochy místnosti. Právě tato vlastnost je něco, co ostatní řešení zmíněné v sekci 4.4.5 neposkytují. Výsledný produkt by tedy měl sloužit pro rozsáhlejší vizualizace, podobné virtuálním systémům popsaných v sekci 2.2. Systém je navržen pro systémy s jednou kamerou, bez použití hloubkových čidel či gyroskopů, okluze virtuálních objektů tedy nejsou řešeny.

Pro implementaci byla využita knihovna ARToolKit. Zaměření této knihovny na použití markerů se ukázalo být robustnější volbou než bezmarkerové systémy, kde dochází k „driftům“, tedy znehodnocení mapování prostoru. Jako vizualizační nástroj byl použit herní engine Unity, pro který ARToolKit poskytuje plug-in, který bude popsán dále v textu. V následujících sekcích se budeme zabývat popisem postupů a praktik, které byly při vývoji použity. Implementaci přecházelo měření vlastností markerů, se kterými ARToolKit pracuje.

Obrázek 11: Nástin řešené problematiky, červeně jsou vyznačeny virtuální objekty, které jsou sledovatelné díky množině markerů rozmístěných v prostoru.

7 Měření detekce markerů

Před samotnou implementací bylo žádoucí vědět, jaké limity a možnosti markerové sledovaní v ARToolKitu obsahuje. Bavíme-li se o vizualizaci v místnostech různých velikostí, pak také velikost zvolených markerů musí být tomuto rozměru uzpůsobena. Provedli jsme tedy sadu měření pro zjištění přesnosti markerů a jejich maximální a efektivní vzdálenosti.

7.1 Měření přesnosti rozpoznání markerů

Bylo testováno, zda knihovna ARToolkit dokáže správně rozpoznat různé vzory markerů. Testování proběhlo na markerech velikosti 5,5×5,5 cm ve vzdálenosti kolem 4 m. Čtyřmetrová hranice byla zvolena na základě předchozího měření, které ukázalo, že právě tato hodnota je maximem pro správnou detekci.

Na kameru bylo postupně vystavena sekvence pěti různých markerů. Čtyři, jež měl program správně identifikovat a jeden, jež nebyl nijak nastaven a sloužil ke „zmatení“ rozpoznání. Sledovali jsme 4 veličiny:

- True Positive (TP) - správně identifikovaný marker
- True Negative (TN) - správné zamítnutí rozpoznání
- False Positive (FP) - nesprávné identifikování markeru
- False Negative (FN) - nerozpoznání markeru

Bylo provedeno celkem 25 pokusů, měření ukázalo následující hodnoty těchto veličin:

Veličina	Počet výskytů
TP	22
TN	11
FP	2
FN	0

Tabulka 1: Výsledky měření veličin rozpoznání markerů

Z výsledných hodnot lze spočítat přesnost rozpoznání dle vzorce

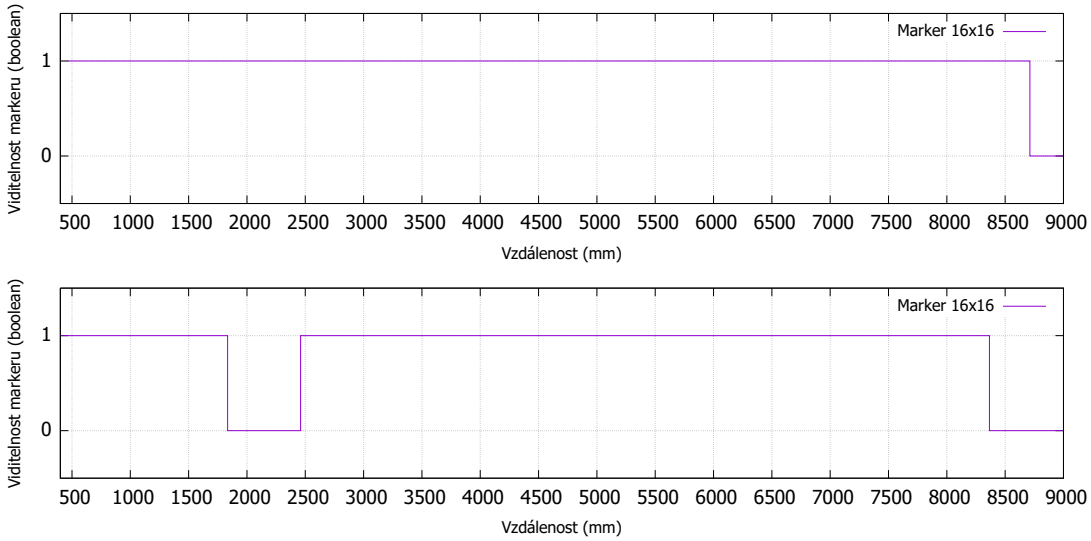
$$Acc = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{1}$$

Výsledná přesnost rozpoznání markeru velikosti 5,5 cm^2 knihovnou ARToolkit v extrémní vzdálenosti je tedy 94,3% za příznivých světelných podmínek.

7.2 Měření detekovatelnosti markerů v závislosti na vzdálenosti

Pro zjištění spolehlivosti detekce bylo provedeno několik měření. Hlavním cílem bylo zjistit do jaké vzdálenosti je ARToolkit spolehlivě schopen sledovat daný marker a z jaké vzdálenosti jej dokáže detekovat. Měření proběhlo v rozmezí 30 cm až 9 m.

Měření markeru velikosti 16×16 cm Z grafů 12 můžeme vidět, že pro největší typ markeru nenastávají v detekci problémy téměř až do samotné maximální vzdálenosti měření. Tato velikost neprokázala vlastnost přerušované detekce (blikání) a po překročení vzdálenosti kolem 8,7 m marker již nebylo možno detekovat.

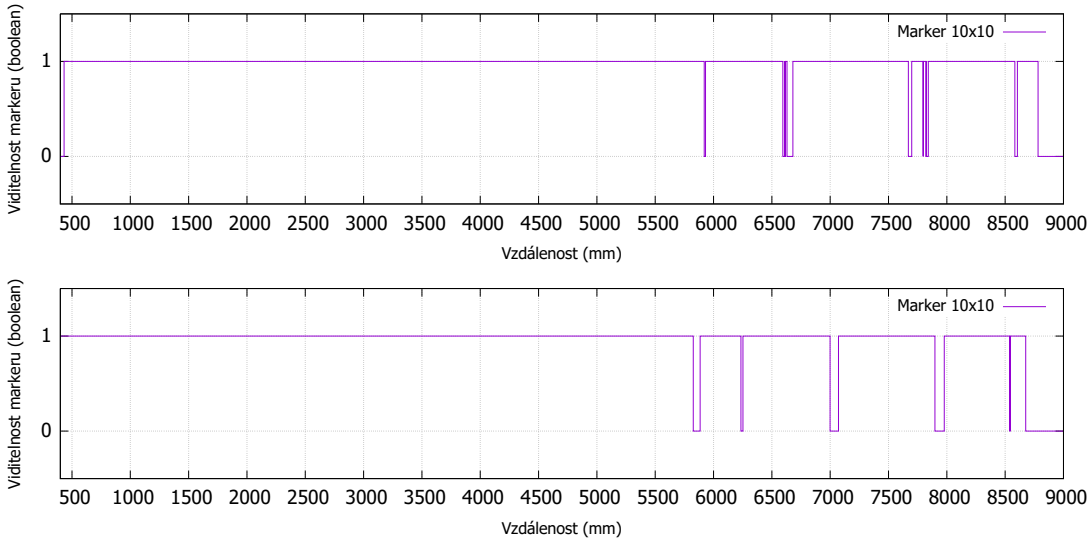


Obrázek 12: Graf sledování viditelnosti markeru (16 cm^2) se zvyšující (nahoře) a snižující se (dole) vzdáleností

To samé měření bylo provedeno i z druhé strany - tedy místo vzdalujícího se markeru byl sledován marker se přibližující. Cílem tohoto pokusu bylo zjistit, z jaké vzdálenosti je knihovna ARToolkit schopna markery zaměřit. Zde můžeme pozorovat, že marker byl nalezen ve vzdálenosti kolem 8,4 m. Chyba v detekci kolem 2 m byla způsobena špatným osvětlením (odlesk světla na papíře se markerem).

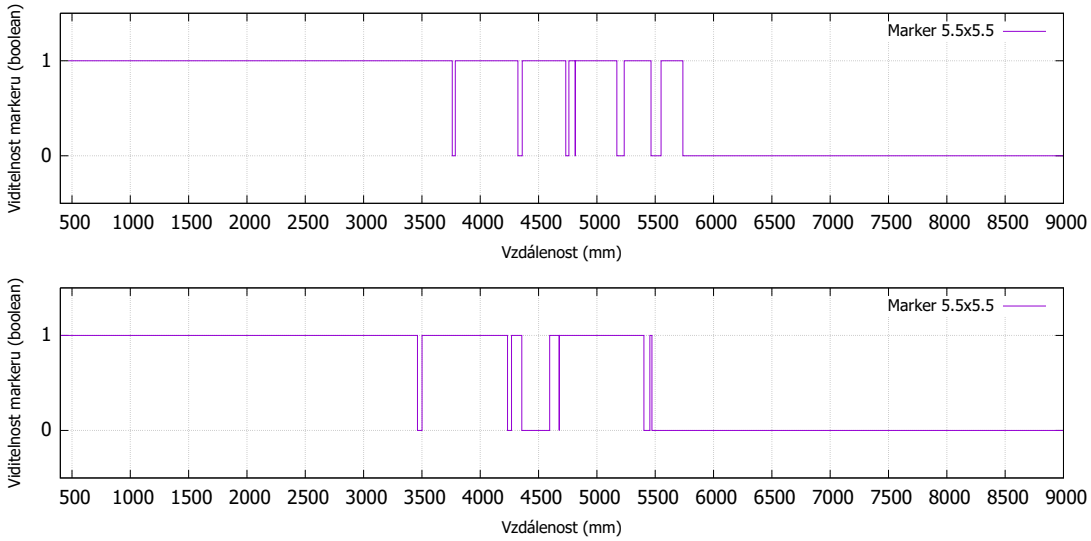
Měření markeru velikosti 10×10 cm Na grafech 13 lze pozorovat velmi přesné sledování markeru do vzdálenosti cca 5,8 m. Za touto hodnotou dochází k výpadkům v detekci, a spolehlivost tak klesá. Po překročení vzdálenosti 8,7 m již nebylo marker možno detekovat vůbec.

Při sledování opačného pohybu můžeme vidět, že marker je zachycen kolem 8,5 m, k ustálení detekce ovšem dojde až kolem 6 m vzdálenosti. Můžeme si všimnout, že tyto hodnoty blízce korespondují s naměřenými hodnotami v předchozím pokusu.



Obrázek 13: Graf sledování viditelnosti markeru (10 cm) se zvyšující (nahoře) a snižující se (dole) vzdáleností

Měření markeru velikosti 5,5×5,5 cm Nejmenší měřená velikost (obr. 14) markeru se projevila dobrou spolehlivostí do vzdálenosti cca 3,6 m. Marker již nebylo možno nalézt po překročení hranice 5,7 m. Oproti největšímu markeru můžeme sledovat přerušovanou detekci mezi těmito dvěma hodnotami.



Obrázek 14: Graf sledování viditelnosti markeru (5,5 cm^2) se zvyšující (nahoře) a snižující se (dole) vzdáleností

V druhém měření byly zjištěny podobné hodnoty, marker byl nalezen ve vzdálenosti 5,5 m, ale k ustálení došlo až kolem 3,5 m. Stejně jako v předchozím měření dochází mezi těmito vzdálenostmi k přerušení sledování.

7.3 Měření míry chvění markeru v závislosti na vzdálenosti

Při prvotním testování knihovny bylo zpozorováno mírné chvění, jehož intenzita se stupňovala, když jsme zvyšovali vzdálenost kamery od markeru. Následující měření má za cíl zjistit efektivní vzdálenost, ze které lze marker spolehlivě sledovat s minimální chybou. Byly celkem provedeny dvě měření, jedno pro „up“ vektor (ve směru osy Z) a druhé pro vektor „forward“ (ve směru osy X). Míra chvění vypočteme jako úhel mezi dvěma korespondujícími vektory ve dvou po sobě jdoucích rámcích, tedy:

$$Jm_{up}^{n+1} = \arccos(m_{up}^{n} \cdot m_{up}^{n+1}) \,, \tag{2}$$

$$Jm_{forward}^{n+1} = \arccos(m_{forward}^{n} \cdot m_{forward}^{n+1}) \,, \tag{3}$$

kde Jm_{up}^{n+1} a $Jm_{forward}^{n+1}$ jsou úhly, o které se dané vektory posunuly vzhledem k předchozímu rámci. Vektory v n-tém rámci jsou označeny m_{up}^{n} a $m_{forward}^{n}$, v následujícím rámci pak m_{up}^{n+1} a $m_{forward}^{n+1}$. Měřilo se celkem v šesti vzdálenostech po dobu cca 20 sekund. V tabulce 2 jsou zobrazeny výsledky měření pro „up“ vektor. V tabulce 3 jsou pak zobrazeny výsledky měření pro „forward“ vektor.

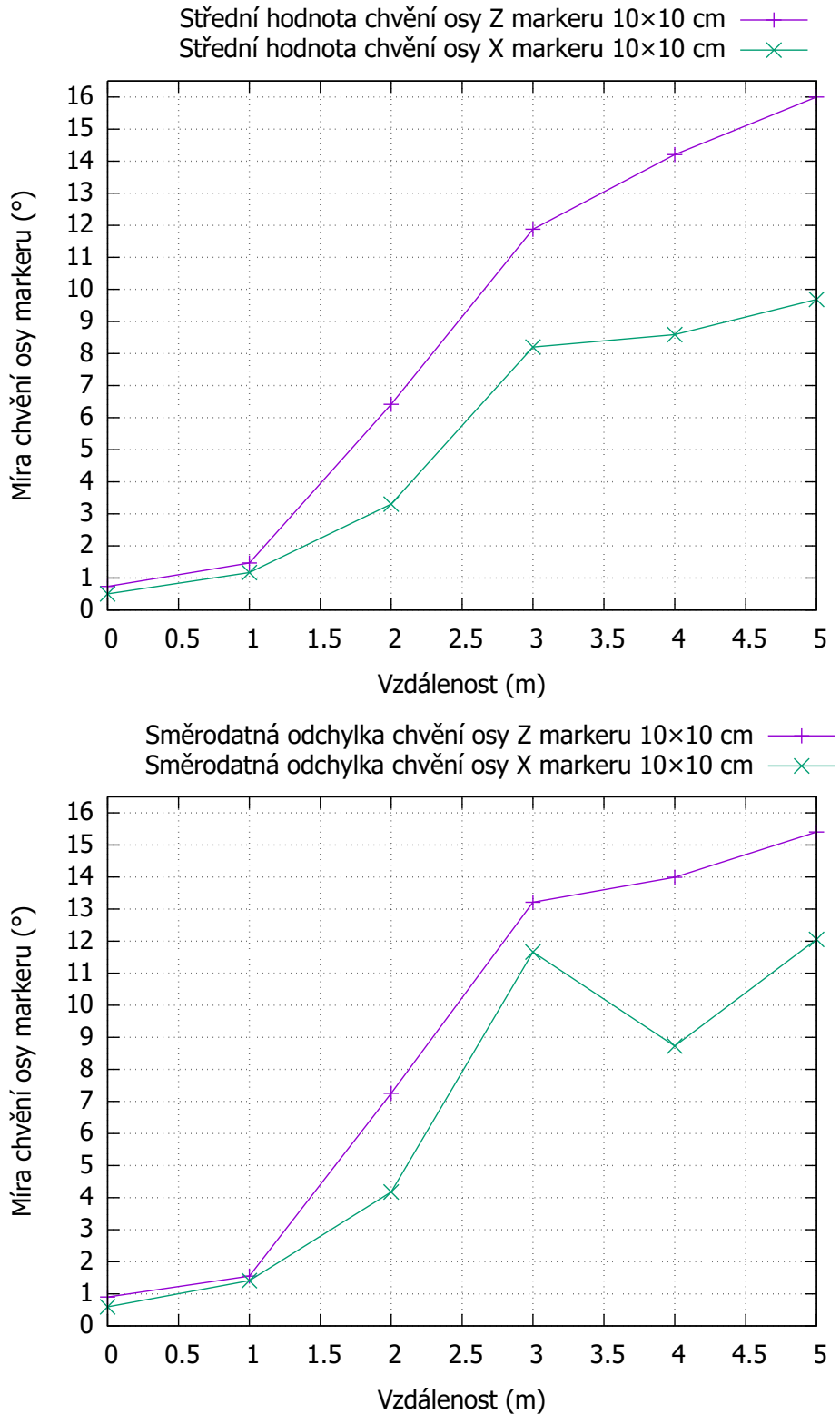
Vzdálenost (m)	Střední hodnota (°)	Min (°)	Max (°)	Směrodatná odchylka (°)
0,2	0,734	0,155	1,840	0,902
1	1,466	0,237	3,110	1,551
2	6,416	0,236	14,426	7,253
3	11,875	0,876	30,858	13,212
4	14,205	1,272	25,095	13,992
5	16,000	1,575	31,113	15,403

Tabulka 2: Výsledky měření míry chvění markeru 10cm^2 v ose Z

Vzdálenost (m)	Střední hodnota (°)	Min (°)	Max (°)	Směrodatná odchylka (°)
0,2	0,504	0,020	1,205	0,592
1	1,167	0,086	2,916	1,412
2	3,301	0,077	9,726	4,174
3	8,199	0,099	29,170	11,659
4	8,589	0,540	23,358	8,728
5	9,687	0,451	28,353	12,054

Tabulka 3: Výsledky měření míry chvění markeru 10cm^2 v ose X

Považujeme-li za nízkou hodnotu chvění hodnoty menší než 5-7°, pak z naměřených hodnot a grafů na obrázku 15 zjistíme, že celková efektivní vzdálenost se prakticky zkrátí na polovinu.



Obrázek 15: Grafy měření míry chvění markeru, nahoře můžeme vidět střední hodnoty v závislosti na vzdálenosti markeru, dole pak směrodatné odchylky.

8 Implementace aplikace v Unity

Pro implementaci AR do prezentační aplikace byla zvolena verze ARToolKitu s připraveným plug-inem pro Unity. Unity je herní engine společnosti Unity Technologies se sídlem v San Franciscu. Tento nástroj umožňuje vytvářet 2D, 3D, ale i VR a AR aplikace s velmi jednoduchou převoditelností na různé platformy.

Herní engine v našem případě slouží jako systém pro renderování celé virtuální scény. V rozhraní Unity rozpoznáváme několik základních struktur:

- GameObject - jakýkoliv objekt ve scéně. Ze základu je tato struktura prázdná až na komponentu Transform, umožňující umístění do prostoru. Funkci mu dávají až přiřazené komponenty.

- Component - třída s logikou, která přiřazenému objektu dává příslušnou funkcionalitu a vlastnosti. Chceme-li například vytvořit ve scéně světlo, pak k prázdnému GameObjectu přiřadíme komponentu Light.

- Prefab - struktura, která dokáže uložit hierarchii GameObjectů i s jejich přiřazenými komponentami.

Skripty s logikou objektů je v Unity možno psát v jazycích JavaScript a C#, pro potřeby této práce byla zvolena druhá možnost, C#. Plug-in ARToolKit pro Unity poskytuje balík vlastních základních skriptů zaobalující jeho knihovnu psanou v jazyce C++.

8.1 Funcionalita ARToolKitu v Unity

V základním balíku skriptů je přes deset pomocných souborů, zde si popíšeme hlavně čtyři nejdůležitější, které byly při tvorbě práce využity. Později na těchto souborech bude stavěna logika prezentační aplikace.

AR Controller Kontrolér, který se celkově stará o běh detekčního plug-inu. Umožňuje nám nastavit základní parametry jako zařízení pro záznam, hodnoty pro prahování, množiny použitých markerů apod. Právě toto poslední zmíněné nastavení je velmi důležité, ARToolKit dokáže pracovat s několika množinami markerů. Ve výsledné aplikaci byly použity množiny dvě: markery s kódem 3×3, kterých může být až 64 a markery 3×3 s Hammingovskou vzdáleností 3, tzn. markerů je méně (přesněji 8), ale snižuje se šance nesprávné detekce.

AR Marker Tento skript drží veškeré informace o markeru. Mezi nejdůležitější vlastnosti v našem případě patří identifikátor *Barcode ID*, *Marker Tag*, šířka markeru a informace o filtraci. Filtrace nám umožňuje redukovat třes markeru vznikly chybou záznamového zařízení a dalších vlivů. V této třídě se také nachází transformační matice daného markeru, díky které můžeme určovat polohu v prostoru.

AR TrackedObject Ke každému markeru můžeme přiřadit scénu, jejíž poloha bude záviset na poloze tohoto markeru. Scénu chápeme jako hierarchii GameObjectů, např. modely. Spárování provedeme aplikací tohoto skriptu na nejvyšší prvek hierarchie. Hlavním parametrem zde je *Marker Tag* - název markeru, shodný s tím v komponentě AR Marker. Ve funkci *LateUpdate* získáváme transformační matice bázového a přiřazeného markeru s jejichž pomocí jsme schopni vypočítat jeho relativní polohu a rotaci vůči středu prostoru. Zde je řešena také viditelnost dané scény, pokud fyzický marker nelze detekovat, pak je skryt také virtuální obsah.

Tato třída také poskytuje tři události:

- onMarkerFound - vyvolaná nalezením registrovaného markeru,
- onMarkerTracked - spouštěna při každé aktualizaci obrazu (frame), pokud je daný marker viditelný,
- onMarkerLost - volaná při ztrátě detekce markeru.

AR Origin Objekt s přiřazenou komponentou AR Origin reprezentuje střed prostoru rozšířené reality, tzv. *World space*. Další důležitou funkcí tohoto skriptu je správa všech markerů ve scéně, drží si jejich reference a dokáže z nich vybrat bázový marker (takový, který leží ve středu world space).

8.2 Základní scéna s ARToolKit

Struktura základní scény (zmíněné v sekci 4.1) s knihovnou ARToolkit vypadá následovně:

- ARToolKit [EmptyObject] (ARController, ARMarker)
- Scene Root [EmptyObject] (AROrigin)
    - Directional Light [DirectionalLight]
    - Camera [Camera] (ARCamera)
    - Marker Scene [EmptyObject] (ARTrackedObject)
        - Cube [Cube]

Legenda: Jméno objektu [Typ Objektu] (Přiřazené skripty knihovny)

8.2.1 Nastavení jednotlivých objektů

Pro správné fungování scény musíme nastavit některé hodnoty v komponentách na objektech scény. Většinou se jedná o kalibrační informace a nastavení požadované funkčnosti objektů. V následujících odstavcích popíšeme jednotlivé objekty, jejich komponenty a důležitá nastavení, potřebné pro správnou funkčnost vizualizace.

ARToolKit Objekt pro inicializaci a nastavení markerů. Vrstva (Layer) tohoto objektu bychom měli nastavit na *Default*. Tento asset má dvě komponenty: ARController a ARMarker. Nejdůležitějšími parametry těchto komponent jsou:

- ARController
  - Camera Parameters: je nutno nastavit na název souboru s informacemi o kalibraci kamery
  - Layer: hodnota *default*
- ARMarker
  - Marker Tag: název markeru (libovolný)
  - Type: hodnoty *Square* nebo *Barcode*, dle vybraného markeru
  - Pattern file: hodnota *patt.hiro* (v případě typu *Square*)

Scene Root Objekt držící celou scénu. Vrstva tohoto objektu (a všech jeho potomků) by měla být nastavena na *AR Background*. Tento asset obsahuje jednu komponentu, kterou není třeba dále nastavovat.

Camera Objekt kamery. Musíme nastavit parametr *Culling mask* na *AR Background*. Nastavení dalších komponent není třeba upravovat.

Marker Scene Scéna pro marker. Potomci tohoto objektu se zobrazí ve výsledném obraze. Tento asset obsahuje jednu komponentu. Tato komponenta má parametr *Marker Tag*, který musíme nastavit na název dříve definovaného markeru.

8.3 Markerové pole

Většina řešení interiérové vizualizace v AR předpokládá, že uživatel bude zobrazovat modely nábytku v omezeném prostoru, většinou přímo na markeru (viz sekce 4.4.5, Ikea). Pokud chceme zobrazit další nábytek, musíme přidat další marker. Toto řešení zapříčiní, že vizualizace každého modelu je přímo závislá na viditelnosti přiřazeného markeru. Při zobrazení celé scény se tedy může stát, že polovina objektů nebude vůbec přítomna.

Takto zobrazená scéna je vhodná pro prezentaci jednotlivých kusů nábytku z katalogu, ale chceme-li provádět vizualizaci nad větší plochou, například nad celým pokojem, je třeba využít pokročilejších technik pro zajištění dostatečného pokrytí. Máme-li tedy jednu místnost, ve které chceme vizualizovat, nabízí se možnost ji pokrýt dostatečným počtem dobře viditelných markerů, přičemž budeme znát jejich pozice. Takové řešení by znamenalo, že pokud můžeme v systému detekovat alespoň jeden z dané množiny markerů, pak také můžeme určit pozici kamery vůči všem ostatním a není tak problém zobrazit celou scénu.

Hlavním cílem zde tedy je vybudování sítě markerů se společným přidruženým prostorem, ve kterém se bude nacházet vizualizovaná scéna. Požadovanou vlastností této sítě je, abychom z jednoho markeru byli schopni získat pozice ostatních. Pro tento účel byla využita metoda popsaná v [31].

V prvním kroku této metody se snažíme v reálném čase získat z obrazu všechny unikátní dvojice zároveň viditelných markerů, mezi nimiž budeme hledat matici, která popisuje relativní transformaci z jednoho markeru do druhého. K tomuto výpočtu potřebujeme znát transformační matice markerů relativních vůči kameře. ARToolKit pro Unity tuto informaci uchovává v každé instanci třídy ARMarker. Tento výpočet provádíme pouze při první společné detekci obou markerů. Matici popisující relativní transformaci $T_{m_1 m_2}$ získáme ze vzorce:

$$T_{m_1 m_2} = T_{m_1}^{-1} T_{m_2} \ , \tag{4}$$

kde T_{m_1} je transformační matice prvního markeru a T_{m_2} náleží markeru druhému.

Společně s touto hodnotou je vhod uložit si informace o spolehlivosti detekce obou markerů, tedy číselné vyjádření, které nám řekne, s jakou jistotou byly dané markery detekovány. Do Unity ARToolKit bohužel takovou informaci nepropaguje. Za hodnotu, která spolehlivost v tomto případě nahrazuje, byl zvolen úhel mezi dopředným vektorem kamery a vektorem markeru směřujícím nahoru. Tímto způsobem získáme informaci, zda je marker vůči kameře kolmo, nebo na něj kamera nahlíží pod úhlem. Výpočet dle vzorce:

$$rel_m = c_{forward} \cdot m_{up} \ , \tag{5}$$

kde rel_m je výsledný index spolehlivosti, $c_{forward}$ je vektor kamery směřující kupředu a m_{up} je vektor markeru směřující nahoru. Takto získané indexy obou markerů můžeme například zprůměrovat nebo vybrat menší z nich a výsledek přiřadit k dané matici. V této práci byla zvolena první metoda.

Detekujeme-li dvojici, kterou již máme uloženou, můžeme využít nových informací ke zpřesnění této relativní transformace. Přesněji využijeme kontinuální vážený průměr obou transformačních matic, kde váha je spolehlivost dané transformace. Pro N-tou detekci vzorec pro aktualizaci relativní transformační matice vypadá následovně:

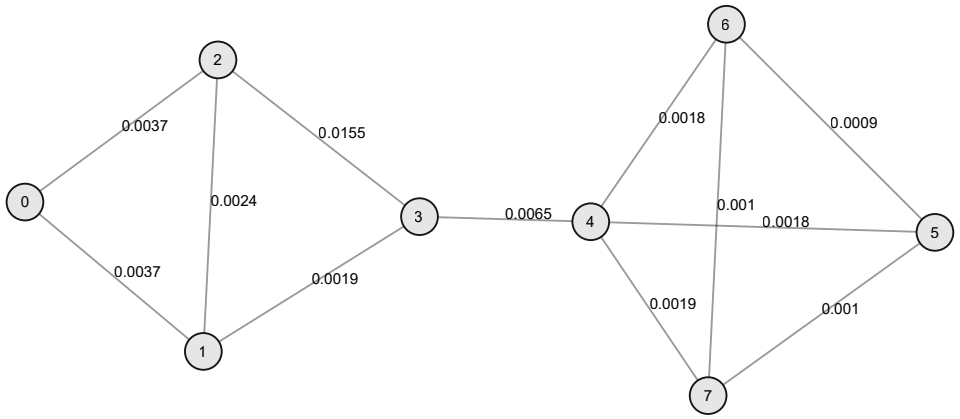
$$T_{m_1 m_2} = \frac{\sum_{i=1}^{N-1} c_{12}^i}{\sum_{i=1}^{N} c_{12}^i} T_{m_1 m_2}^{N-1} + \frac{c_{12}^N}{\sum_{i=1}^{N} c_{12}^i} T_{m_1 m_2}^{N} \ . \tag{6}$$

Jakmile máme dostatečný počet takto popsaných vazeb, můžeme je uspořádat do grafu, kde množinou vrcholů je množina markerů a hrany reprezentují právě tyto vazby. Takové uspořádání nám umožní sledovat jednotlivé návaznosti markerů, to znamená, vezmeme-li v potaz příklad na obrázku 16, že pokud víme, kde se v reálném prostoru nachází marker č. 0, pak od něj lze dohledat pozici markeru č. 3. Jak lze z příkladu vidět, vedou k němu dvě cesty: jedna přes marker č. 1, druhá přes číslo 2. Pro volbu nejlepší cesty byl využit Dijkstrův algoritmus. Jednou

z možností ocenění hran je využití spolehlivosti markeru získaného z rovnice 5. Jelikož hledáme nejkratší cestu a tento index určuje spolehlivost, tudíž větší hodnoty příznivé pro nás, ale pro hledání cesty je tomu naopak, výsledné ohodnocení získáme odečtením indexu od jedné:

$$edge_{m_1 m_2} = 1 - rel_{m_1 m_2} \ . \tag{7}$$

Tento princip ocenění hran má za cíl vybrat cestu takovou, které bylo dosaženo s co nejlepší možnou viditelností všech markerů a tudíž jsou jejich dané transformační matice nejpřesnější. Jako alternativní možnost je cenu hrany ponechat konstantní (např. 1) a nejkratší cesta bude tedy ta s co nejmenším počtem přeskoků.



Obrázek 16: Příklad grafové reprezentace markerového pole. Uzly grafu odpovídají markerům s daným ID, hodnoty hran určují spolehlivost společné detekce.

V síti jeden marker zvolíme jako tzv. bázový (base), vůči kterému se budou počítat pozice ostatních markerů. Tento marker je přímo navázán na tzv. origin (počátek) virtuální scény - souřadnice (0, 0, 0). Při ztrátě detekce bázového markeru se vybere z viditelných markerů nový a vůči němu se přepočtou nejkratší cesty. Volba toho markeru může být libovolná, v našem případě je vždy zvolen marker s nejnižším identifikačním číslem. Samotné umístění tohoto markeru v prostoru je velmi důležité, jeho umístěním do středu sítě zabráníme dlouhým cestám a celkově tak zpřesníme samotný výpočet.

Systém markerového pole je nyní kompletní. Pozice markerů v této síti nemusí být předem známy, markery mohou být položeny v libovolných úhlech a pozicích. Jedinou podmínkou pro správné fungování tohoto systému je, že se markery nebudou pohybovat a celá síť zůstane sta-

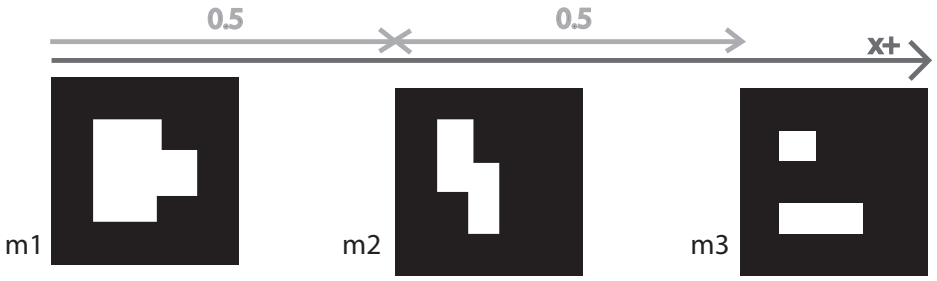
tická. Zdokonalili jsme tak metodu popsanou v počátku této sekce, ztráta detekce markeru již neznamená ztrátu celé sítě. Transformaci z báze do markeru vypočítáme dle vzorce:

$$T_{bx} = T_{bm_1} T_{m_1 m_2} T_{m_2 m_3} \dots T_{m_n m_x} \ , \tag{8}$$

kde T_{bx} je požadovaná transformace a $T_{m_x m_y}$ jsou transformační matice vazeb na nejkratší cestě od bázového ke hledanému markeru.

8.3.1 Příklad využití markerového pole pro nalezení markeru

Vezměme si příklad se třemi markery v prostoru, které jsou pozicované v řadě s rozestupy 50 cm. Marker m_1 je v markerovém poli přímo spojen s markerem m_2, jež má vazbu na marker m_3. Tento případ znázorňuje obrázek 17. Demonstrujme si nyní postup výpočtů, které se provádí při inicializaci a používání markerového pole pro nalezení pozice a pózy markeru, jež není možno detekovat.



$$T_{m_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} T_{m_2} = \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} T_{m_3} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obrázek 17: Zjednodušený příklad situace, kdy tři markery leží vedle sebe ve směru osy X s rozestupy 50 cm. Tm_1, Tm_2, Tm_3 jsou transformační matice daných markerů v prostoru kamery.

Marker m_1 je v našem případě bázový. Při tvorbě sítě byla vypočtena relativní transformační matice $T_{m_1 m_2}$ s využitím vzorce 4. Příklad výpočtu v našem případě můžeme vidět ve vzorci 9.

$$
\begin{aligned}
T_{m_1m_2} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{9}
$$

Stejný postup byl využit pro výpočet transformace z markeru m2 do markeru m3. Výsledná matice $T_{m_2m_3}$ byla vypočtena následovně:

$$
\begin{aligned}
T_{m_2m_3} &= \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0,5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{10}
$$

Představme si nyní situaci, kdy při sledování kamery ztratíme z pohledu marker m_3 a je nutno jeho pozici zjistit z markerového pole. Nejkratší cesta v našem případě od bázového vektoru je $m_1 \rightarrow m_2 \ \rightarrow m_3$. V grafu tedy vyhledáme jednotlivé transformační matice a s použitím vzorce 8 transformaci z počátku získáme následovně:

$$
T_{m_1m_3} = T_{m_1m_2}T_{m_2m_3} = \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0,5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}
$$

Takto získaná transformační matice nám říká, že od bázového markeru se marker $m3$ nachází 1 m ve směru osy X, přesně jak bylo stanoveno v zadání problému.

8.4 Implemetace markerového pole v Unity

Pro implementaci výše zmíněných metod jsme museli pozměnit funkčnost základních tříd, které ARToolKit pro Unity poskytuje. Do scény tak mimo jiné přibyly třídy *ARMarkerField*, *ARMarkerFieldOrigin*, *ARTrackedFieldObject* a *Graph*, které si nyní podrobně rozebereme.

8.4.1 ARMarkerFieldOrigin

Tato třída dědí z AROrigin a rozšiřuje jeho funkci výběru bázového markeru o možnost prioritizovat marker uživatelem vybraný.

GetBaseMarker *()* : ARMarker - metoda, která ze seznamu všech výchozích markerů vybere bázový, přičemž prioritizuje marker, který je bázovým v markerovém poli. Pokud takový není schopen najít, pak ze seznamu vybere jiný viditelný marker. Při změně bázového vektoru je proveden přepočet nejkratších cest v grafu.

8.4.2 Graph

Graph je třída implementující datovou strukturu graf, která drží veškeré informace o vazbách markerů. Struktura je realizovaná jako matice sousednosti. Místo hodnot hran je do ni však ukládána instance třídy EdgeData, která obsahuje jak tuto hodnotu, tak dané transformační matice. Tato třída také disponuje metodou pro aktualizaci této transformace podle vzorce 6. Nejdůležitějšími třídními proměnnými jsou tedy tyto:

- EdgeData[][] edges - matice sousednosti, dvourozměrné pole objektů EdgeData;
- int originNode - číslo ID uzlu, jež zrovna figuruje jako báze a k němuž se počítají nejkratší cesty;
- int[] parentNodes - pole obsahující strom nejkratších cest k bázovému markeru.

Z metod této třídy zmíníme tyto čtyři nejdůležitější metody:

AddEdge *(int id1, int id2, ARMarker m1, ARMarker m2, float value)* : void - metoda pro inicializaci nové hrany. Zde se vytvoří instance třídy EdgeData, do které se vloží její ohodnocení a vypočítá se transformační matice z markeru s id1 do markeru s id2 dle vzorce 4. Je zde také vytvořena opačná hrana se stejným ohodnocením, ale inverzní maticí pro transformaci.

GetShortestPaths *()* : int[] - metoda provede Dijkstrův algoritmus nad grafem. Výsledkem je strom nejkratších cest, který je uložen do proměnné *parentNodes*. Volá se vždy, pokud se graf změní, případně se změní uzel figurující jako bázový marker.

DecodeShortestPaths ***(int from)*** **: int**[] - získá nejkratší cestu k *originNode* z uzlu *from*. Výsledkem je seznam identifikátorů všech uzlů po nejkratší cestě seřazených od bázového do výchozího.

SetOrigin ***(int id)*** **: void** - nastaví uzel, k němuž se budou nově vypočítávat nejkratší cesty. Tyto jsou znovu přepočteny.

8.4.3 ARMarkerField

Tato třída je určena pro inicializaci, údržbu a následné zrušení markerového pole. Veškerá logika popsaná v sekci 8.3 se nachází zde. Dále jsou na tomto místě drženy informace o markerech, které jej tvoří a jejich přiřazené scény. Třída je použita jako komponenta objektu *ARToolKit* (objekt, který drží ARController) a má tři hlavní parametry:

- Marker Size - velikost markerů tvořící pole (v metrech);
- Number of Markers - počet markerů tvořící pole;
- Origin Marker ID - ID markeru, který má být preferován jako bázový.

Dále třída disponuje celkem devíti důležitými metodami:

Initialize ***(List <int>)*** **: void** - metoda přijímá seznam ID markerů, jež tvoří síť. Následně vytvoří instanci třídy *Graph* s danými uzly. V dalším kroku pro každý marker v seznamu vytvoří komponentu ARMarker, jíž jsou správně nastaveny hodnoty BarcodeID, Pattern Width a Tag. Ke každému markeru je následně vytvořena a přiřazena instance komponenty *ARTrackedFieldObject*.

EnableMarkerField ***(List <int>)*** **: void** - metoda slouží pro aktivaci markerového pole. Základní AROrigin je zda nahrazen instancí ARMarkerFieldOrigin s požadovanými parametry a pole je aktivováno metodou InitializeMF. Objektům scén jsou přiřazeny tzv. *ColliderPlane* (viz 8.6) a v případě bázového markeru také *ShadowPlane* (rovinu, na niž se promítají stíny objektů).

EnableObjectMarkerField ***(List <int>, ShowcaseExample)*** **: void** - stejná funkce jako metoda výše, pouze aktivuje pole pro potřeby objektového mapování. Scéně bázového markeru jsou zde přiřazeny zájmové body popsané v instanci třídy *ShowcaseExample*.

DisableMarkerField ***()*** **: void** - funkce pro deaktivaci markerového pole, volá se zde metoda *Scrap*, která odstraní všechny markery a k nim přiřazené scény. Také se zde aktivuje výchozí AROrigin.

FindMarkerByID *(int)* : ARMarker - tato metoda vrátí instanci ARMarker s příslušným ID.

FindMarkerSceneByID *(int)* : GameObject - metoda, která vrací scénu, jež patří markeru s daným ID.

Update *(ARMarker)* : void - metoda volaná při aktualizaci každého rámce. Prochází všechny dvojice viditelných markerů, které jsou následně zapisovány do grafu, případně, pokud již v grafu existují, provede aktualizaci jejich transformační matice.

GetMarkerPosition *(ARMarker, out Matrix4x4)* : bool - tato metoda provádí výpočet transformační matice popisující přechod z bázového do zvoleného markeru. Z instance třídy Graph je získáno pole popisující tuto nejkratší cestu a jednotlivé hrany grafu. Následně je proveden výpočet dle vzorce 8 a metoda vrací *true* jako návratovou hodnotu. Pokud metoda není schopna najít cestu k danému markeru (graf je nesouvislý), vrací se hodnota *false*.

8.4.4 ARTrackedFieldObject

Poslední zmiňovaná třída je ARTrackedFieldObject. Tato dědí ze třídy ARTrackedObject a mění její princip výpočtu polohy tak, aby se dalo využít v markerovém poli. Původní princip využíval přímé detekce, kdy je pozice vypočtena dle vzorce $P_{m_1} = T_{bm}^{-1} T_{m_1}$ v případě viditelného markeru a scéna markeru, který není vidět je schována. V tomto případě ovšem zamezíme ztrátě scény získáním polohy z markerového pole. Takto ošetřená situace nám umožní zobrazit i objekty, jejichž marker není plně viditelný nebo se nachází mimo záběr. Příklad obou situací je vyobrazen na obrázku 18.

8.5 Přímé zobrazení nábytku

Prvním a také nejjednodušším případem užití je zobrazení jednoho kusu nábytku na jeden marker pomocí přímé detekce. Uživatel má možnost vybrat vzor požadovaného markeru, který má k dispozici. Po zvolení uživatel může ze seznamu vybrat kus nábytku, jež chce zobrazit. Změnou nastavení lze modifikovat vstupní množinu markerů (viz 8.1). Namíříme-li kameru na plně viditelný marker získáme vizualizaci požadovaného modelu.

Pro volbu nábytku bylo vytvořeno dialogové okno, jež v demonstrační aplikaci načítá informace o modelech ze souboru JSON. Jednotlivé prvky je možno seskupovat podle typu a lze jim nastavovat miniaturu pro náhled modelu. Při nasazení ve větších projektech, jako jsou katalogy nebo e-shopy, můžeme použít uložení v databázi.

Obrázek 18: V případě přímé detekce (obrázek nahoře) dojde ke ztrátě přiřazené scény, pokud marker není zcela viditelený. Aplikací markerové sítě (dole) můžeme zobrazit jakoukoliv scénu pokud je alespoň jeden marker viditelný.

8.6 Mapování místnosti

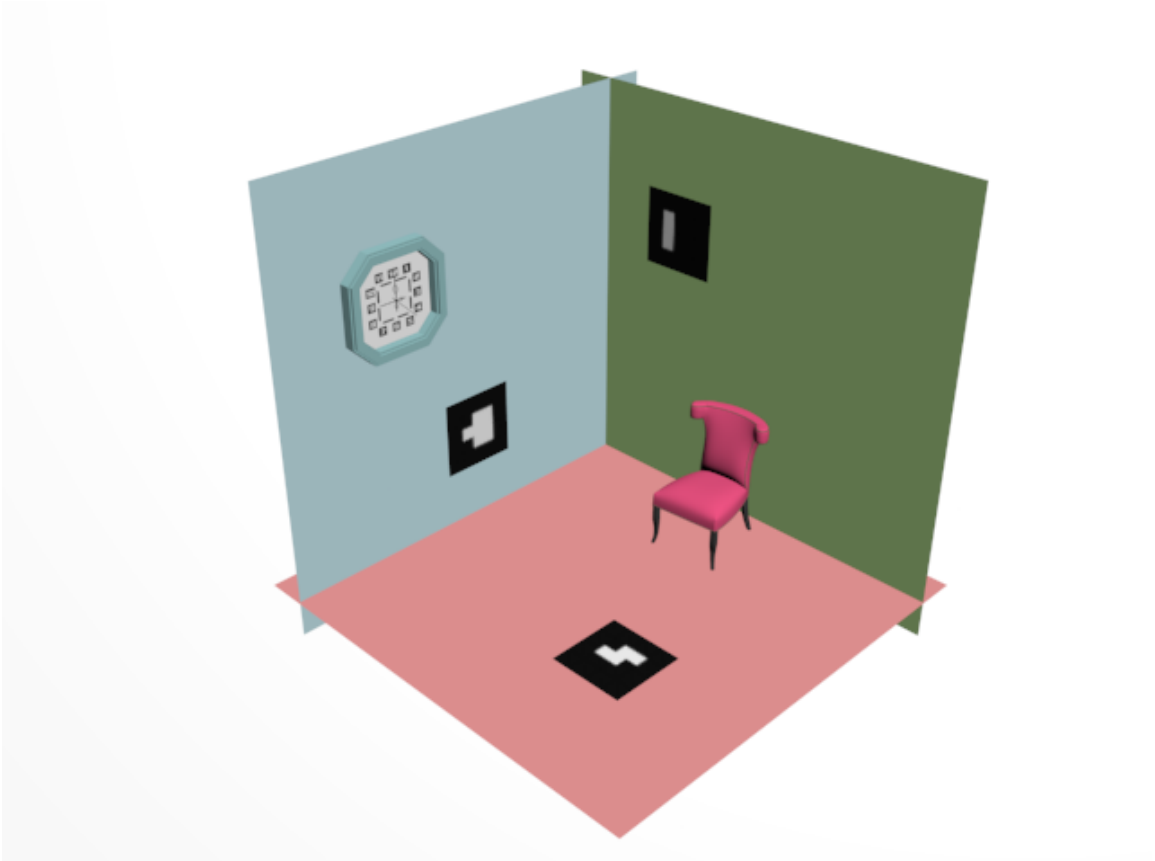
Dalším případem užití rozšířené reality v interiérovém design, který zde demonstrujeme, je rozmístění virtuálního nábytku v místnosti. Pro pokrytí dostatečné plochy využíváme markerové pole. Pro správnou volbu markerů lze využít poznatků ze sekce 7, kdy jsme měřili vhodné velikosti markerů v závislosti na vzdálenosti od kamery a detekce v případě náhledu pod úhlem.

Máme-li místnost o šířce 4 m, délce 7 m a vizualizaci provádíme ze středu této místnosti, pak zjistíme, že pro naše účely jsou vhodné markery o velikosti cca 16 cm^2. V tomto případě se také snažíme minimalizovat šance výskytu negativních pozitiv v detekci, bylo tedy nutno zvážit, zda využít markerové sady s větší Hammingovskou vzdáleností na úkor maximálního počtu markerů.

Uživatel má možnost zvolit z jakých markerů z vybrané sady se síť bude skládat (jaké markery použije pro zmapování místnosti). Z vybraných markerů je ten s nejnižší hodnotou ID zvolen jako bázový a jak bylo doporučeno v sekci 8.3, měl by být umístěn do středu prostoru, ve stejné vzdálenosti k ostatním markerům.

Následně je třeba provést mapování samotného prostoru, kdy je uživatel vyzván k detekci dvojic markerů, aby mohla být vytvořena požadovaná síť. Čím kompletnější je výsledný graf, tím lepší je výsledná detekce. Pozor je třeba si dávat na chyby, které mohou vzniknout nesprávnou manipulací a nastavením kamery, jako je velký motion blur (rozmazání při pohybu), autofokus apod. Takovéto znehodnocení obrazu může mít za následek nesprávnou, případně falešně pozitivní detekci markerů a nefunkčnost celé sítě dokud není znovu aktualizována.

Jakmile je místnost zmapována, uživatel má možnost ze seznamu modelů vybrat požadovaný nábytek a umístit jej dle volby do prostoru. Jelikož již nezobrazujeme jeden model na jediný marker, ale snažíme se vytvořit iluzi reálné místnosti, museli jsme najít jiný způsob, jak tuto funkci umožnit. Výsledný princip je vcelku jednoduchý (viz 19)



Obrázek 19: Obálka místnosti, každým markerem je vedena rovina. Tyto roviny ve výsledku vytvoří hrubý model stěn, vůči kterým lze umisťovat virtuální objekty. V tomro případě můžeme vidět židli, která je přiřazena k rovině podlahy a hodiny, které jsou přiřazeny k modré boční stěně.

Každým markerem vedeme rovinu (v Unity reprezentován objektem Plane), která uživateli není zobrazena, ale je schopna detekovat kolize. Právě tato funkce je využita pro řešení našeho problému pokládání nábytku. Uživatel si vybere požadovaný model a kliknutím do scény jej umístí na požadované místo. V pozadí aplikace při této akci vyšle paprsek z kamery do daného bodu pomocí metody *Physics.Raytrace*, kterou Unity poskytuje. Pro takto vyslaný paprsek se kontrolují kolize s objekty ve scéně. Možnost filtrovat jednotlivé collidery z raytracingu nám dovolí ignorovat vše kromě rovin, které tvoří stěny naší místnosti. Po nalezení kolize získáme všechna potřebná data jako 3D bod v prostoru a normálu roviny z objektu *RaycastHit*. Vybraný model je následně nakloněn vzhledem k rotaci roviny a jeho pozice je nastavena na daný bod průsečíku. Výstup z výsledné aplikace, kde je použit tento postup, je na obrázku 20.

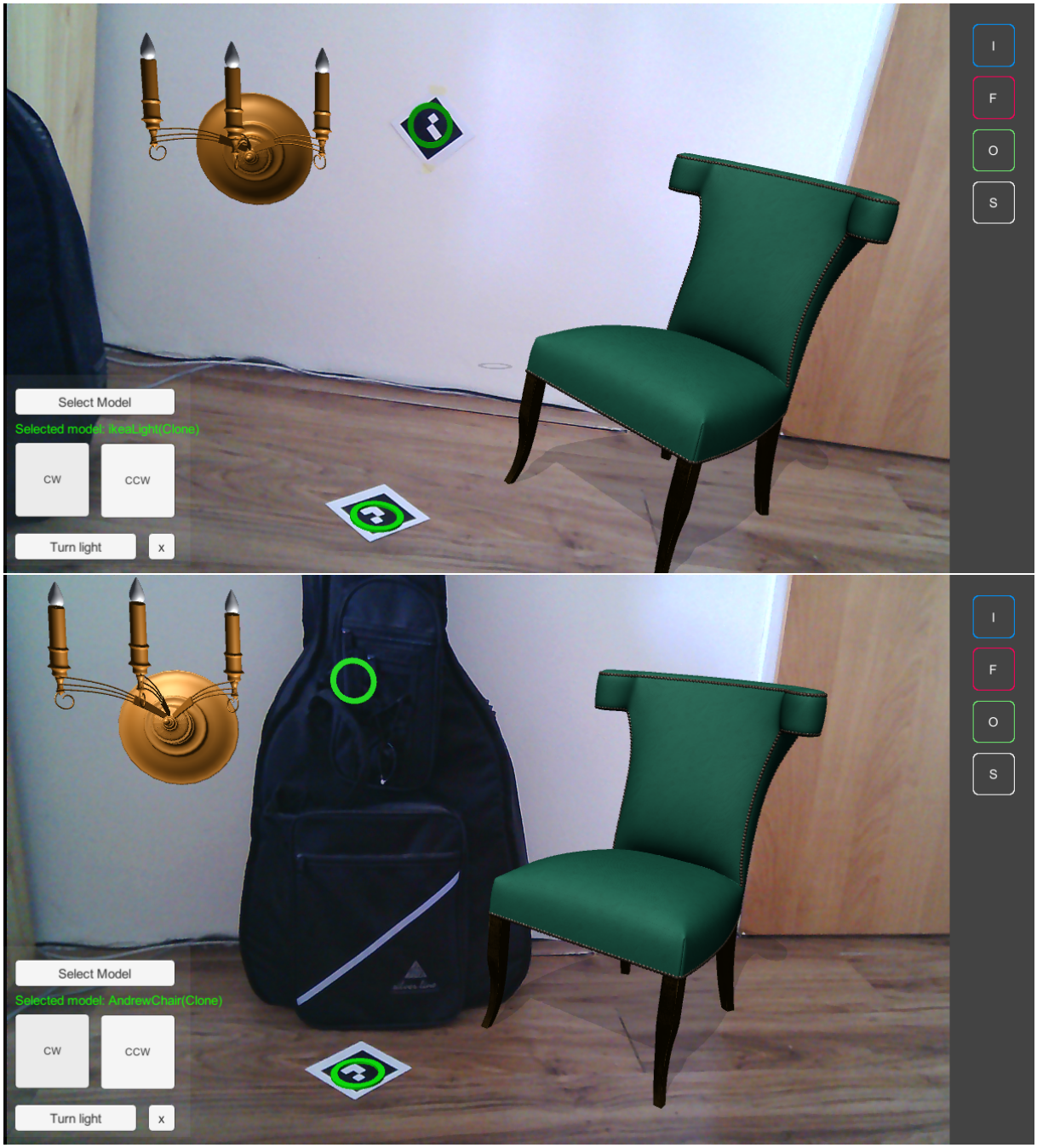
S takto umístěnými objekty nelze manipulovat úpravou pozice markeru, jak tomu bylo v předchozím případě. Pro tento účel tedy bylo zapotřebí mechanismus vybírání, otáčení a odstranění objektů vytvořit. V uživatelském rozhraní byl proto vytvořen jednoduchý modul (obr. 21). Kontrolér tohoto modulu byl nazván *uiRemote* a umožňuje následující operace s objekty ve scéně:

- Výběr objektu - aktivuje se při kliknutí do scény, provede trasování paprsku směrem do scény a protne-li se paprsek s modelem uloží si jeho informace jakožto vybraný (selected) objekt,
- Rotace objektu - uživatel má možnost otáčet vybraným objektem ve směru a proti směru hodinových ručiček stisknutím příslušných tlačítek,
- Odstranění objektu - objekt může být ze scény odstraněn jeho vybráním a stisknutím tlačítka,
- Nastavení směru světla - výběrem této volby má uživatel možnost nastavit orientaci stínů korespondující s reálným prostředím.

8.7 Mapování objektu

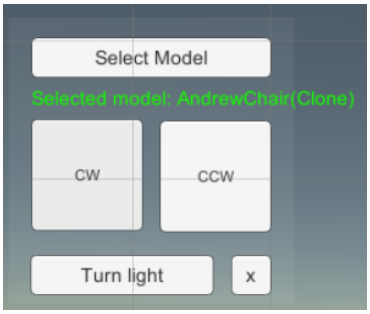
Využití markerového pole není omezeno pouze na použití v místnostech, ale lze jej využít pro mapování objektů. Vezměme si jako příklad kancelářský stůl, na jehož desku chceme umístit objekt. Po programu žádáme, aby byl model stále viditelný, i když kamera přímo nesnímá shora desku stolu a nahlíží na něj z různých úhlů.

Pro řešení tohoto problému bylo využito systému, kdy pozici bázového markeru předem známe (např. roh stolu) a objekt je pozicován vůči němu (obr. 22). Ostatní můžeme osázet po objektu, tyto slouží pro výpočet pozice bázové scény s objektem. Na tyto markery už není kladena žádná další podmínka, a proto mohou být natočeny v libovolných úhlech. Problematika osázení objektu markery zde může působit největší komplikace, chceme-li například, aby byl objekt stále viditelný i při obcházení stolu, pak musíme zajistit, že vždy alespoň jeden marker půjde detekovat.



Obrázek 20: Praktický příklad funkčního mapování místnosti. Virtuální objekty patří do scén markeru na jejichž rovině leží. Na druhém obrázků můžeme pozorovat nepřerušenou vizualizaci lampy i při zakrytí přiřazeného markeru.

Takto zmapovaný objekt může značně usnadnit vizualizaci interiérových prvků. Položíme-li jej do středu místnosti, můžeme zobrazit nábytek v prostoru bez nutnosti někdy složitého a zdlouhavého osázení místnosti markery. Známe li pozici bázového markeru vůči podlaze, můžeme mu přiřadit kolizní rovinu jak tomu bylo v sekci 8.6. Na tuto plochu můžeme zobrazit nábytek



Obrázek 21: Modul uživatelského rozhraní pro manipulaci s objekty.

Obrázek 22: Příklad mapování objektu. Na posledním obrázku si povšimněte absence bázového markeru, scéna je zobrazena výpočtem z markerového pole.

metodou popsanou v té samé sekci. Nevýhodou této metody je nemožnost detekovat zdí.

8.8 Použití interiérových prvků

Interiéry místností mohou obsahovat ovladatelné prvky, jako termostaty, interkomy či jiné ovladače, jejichž ovládání nemusí být zprvu zřejmé. Rozšířené reality zde můžeme využít pro názorné vysvětlení, kdy statické manuály mohou být složité nebo v jazyce, jemuž uživatel nerozumí. Ukažme si princip řešení na příkladu s ovládáním domácího interkomu.

Abychom mohli znázornit jednotlivé akce, které uživatel musí provést, bylo třeba vytvořit sadu ukazatelů a šipek. Tyto prvky byly animovány pro znázornění pohybu. Animace v Unity jsou realizovány pomocí komponenty *Animator*, která trpí vadou, kdy při deaktivaci této komponenty (což nastává při ztrátě markeru) je animace spuštěna znovu od začátku. Výsledkem je nežádoucí přerušovaný efekt.

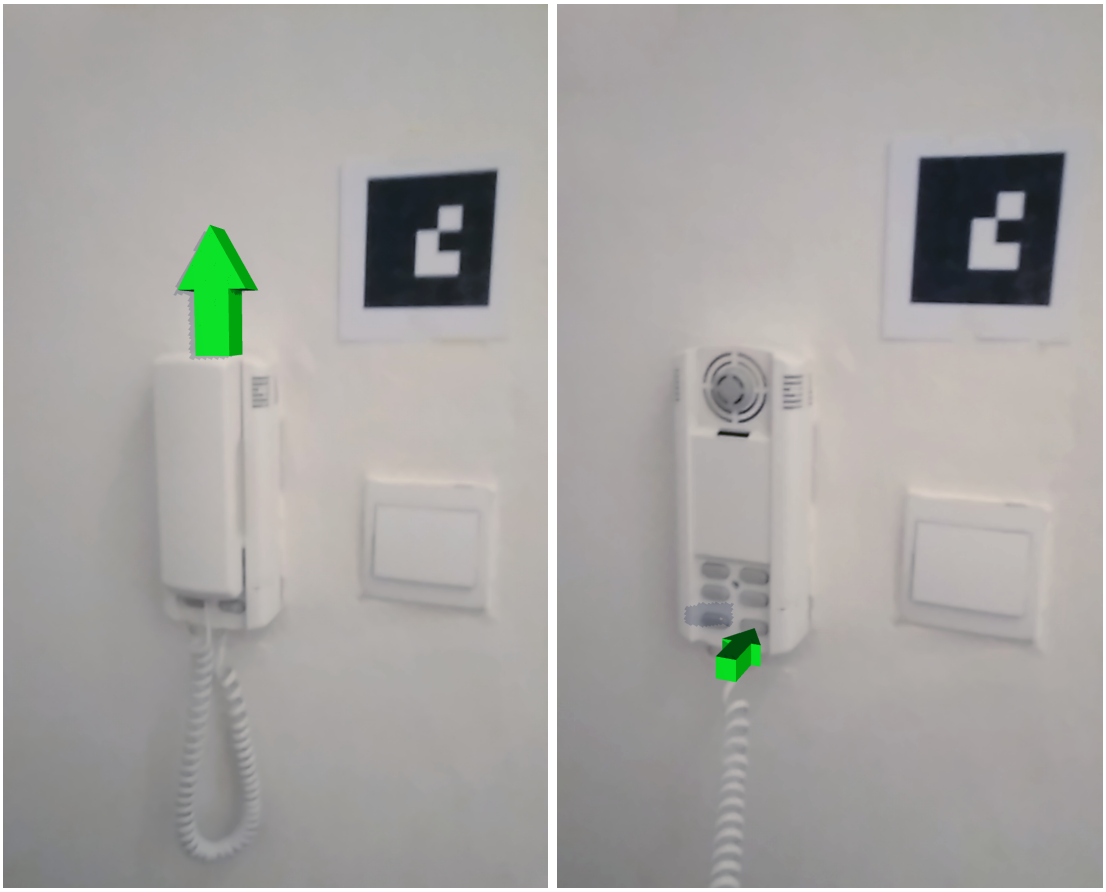
Tento problém jsme vyřešili vytvořením skriptu *animArrow*, který objektu v každém rámci obrazu ukládá aktuální čas animace. Pokud nyní dojde ke ztrátě markeru, známe čas, ze kterého můžeme animaci znovu spustit. To se děje v přetížené funkci *onEnable*. Výsledkem je objekt s plynulou animací i při přerušované nebo nestabilní detekci.

Systém zobrazení těchto notací je stejný jako v předchozím případě s mapováním objektů, tedy je určen jeden bázový marker, který má pevnou polohu a pozice jsou vztaženy vůči němu. Lze zde využít výhod markerové sítě, další markery již mohou mít jakékoliv pozice, slouží pouze k rozšíření mapované oblasti. Při realizaci aplikace bylo využito uložení takovýchto zájmových bodů ve formátu JSON, kdy můžeme nastavit požadovanou pozici a rotaci. Příklad takového uložení lze vidět na výpisu 1.

```
{
   "name": "Intercom use",
   "interestPoints": [
       {
        "position": [ -0.10, -0.10, 0.00 ],
        "rotation": [ 90, 0, 0 ],
        "scale": [ 1, 1, 1 ],
        "prompt": "UIElements/arrowUp"
      },
      {
         "position": [ -0.10, 0, 0.00 ],
         "rotation": [ 0, 0, 0 ],
         "scale": [ 1, 1, 1 ],
         "prompt": "UIElements/arrowUp"
      },
      {
         "position": [ -0.075, -0.2, -0.15 ],
         "rotation": [ 180, 0, 0 ],
         "scale": [ 0.5, 0.5, 0.5 ],
         "prompt": "UIElements/arrowUp"
      }
   ]
}
```

Výpis 1: Uložení zájmových bodů v JSON.

Pokud se daná akce skládá z více kroků, pak je takových zájmových bodů možno nahrát několik a zobrazit v po sobě jdoucí sekvenci. Použitý princip lze vidět na obrázku 23. Pomocí těchto bodů a dostatečného počtu pomocných šipek a notací můžeme docílit názorného popisu vykonání jakékoliv akce.



Obrázek 23: Příklad vizualizace ovládání domácího interkomu, které se sestává ze dvou kroků.

9 Závěr

Cílem práce bylo seznámit se s technikami vizualizace interiérů v rozšířené a virtuální realitě. Popsali jsme výhody a problematiky jednotlivých technologií a jejich využití v různých oborech. U případě virtuální reality byl vysvětlen popis postupu přípravy interiérově scény pro interaktivní procházku v nástroji Unreal Engine. V rámci rozšířené reality jsme se pak zabývali různými přístupy ke sledování kamery v prostoru a metodami pro realistickou vizualizaci, jako například řešení okluzí. V druhé části textu jsme popsali implementaci demonstrační aplikace využívající markerového pole pro rozšíření mapované plochy.

Pro realizaci rozšířené reality jsme použili řešení pomocí markerů s použitím knihovny ARToolKit. Tento systém se vyznačoval výpočetní nenáročností a rychlostí detekce. Jak ovšem bylo zjištěno, markery prokazovaly nepřesnosti v určení pózy se zvyšující se vzdáleností markeru od kamery. Bylo tedy nutno provést měření vlastností jako maximální vzdálenost detekce a míra chvění markeru. Z tohoto měření vyplynulo, že efektivní vzdálenost pro detekci je v cca 50 % vzdálenosti maximální. Jistá nepřesnost v určení pózy bude existovat vždy, způsobená osvětlením prostředí nebo šumem kamery. Tuto vlastnost jsme se snažili minimalizovat filtrací, kterou ARToolKit poskytuje. Tato úspěšně ustálila detekci objektů při statické pozici kamery, způsobila ovšem efekt, kdy virtuální objekt „klouže“ po scéně, než dojde k ustálení. V budoucnu bych zvážil použití prediktivních filtrů, jako např. Kalmanův filtr.

Ve výsledku byl úspěšně realizován způsob, jak interiér osadit virtuálními prvky, bez nutnosti detekovat všechny markery, které jej mapují. Použitý systém se ukázal být vcelku spolehlivý za dobrých světelných podmínek, jakožto všechna markerová řešení ovšem trpí na problémy popsané výše. Pro další práci na této metodě bych zvážil použití hloubkových dat pro řešení okluzí.

Z poznatků zaznamenaných v této práci jsem došel k závěru, že virtuální a rozšířená realita rozhodně najdou ve vizualizaci interiérů využití. Zatímco interiérové studia a designerské firmy s dostatečným rozpočtem, potřebným k přípravě modelů a scény jako takové, sáhnou po virtuální realitě k prezentaci svého produktu, lidé v domácnosti mohou sáhnout po řešeních v rozšířené realitě, která jsou dostupná z mobilních telefonů. Jak jsme si ukázali, takové řešení není omezeno pouze na malé prostory.

Díky této práci jsem se seznámil se zajímavými technikami, technologiemi a nástroji jako Unreal Engine 4, u kterého oceňuji velice výkonný materiálový editor, který značně ulehčuje práci s jinak složitými shadery. Dále pak engine Unity 3D, kde jsem zase ocenil výbornou komunitu a množství dostupné dokumentace.

Literatura

[1] Virtual Reality. *Merriam-Webster* [online]. 2017 [cit. 2017-04-03]. Dostupné z: https://www.merriam-webster.com/dictionary/virtual%20reality

[2] Augmented Reality. *Merriam-Webster* [online]. 2017 [cit. 2017-04-03]. Dostupné z: https://www.merriam-webster.com/dictionary/augmented%20reality

[3] AZUMA, Ronald T. *A Survey of Augmented Reality.* 1997. Hughes Research Laboratories.

[4] MILGRAM, Paul, Haruo TAKEMURA, Akira UTSUMI a Fumio KISHINO. *Augmented Reality: A class of displays on the reality-virtuality continuum.* 1994. ATR Communication Systems Research Laboratories.

[5] The Best VR (Virtual Reality) Headsets of 2017. *PCMag* [online]. 2017 [cit. 2017-04-03]. Dostupné z: http://www.pcmag.com/article/342537/the-best-virtual-reality-vr-headsets

[6] LAVIOLA, Joseph J. *A discussion of cybersickness in virtual environments.* ACM SIGCHI Bulletin [online]. 2000, 32(1), 47-56 [cit. 2017-04-12]. DOI: 10.1145/333329.333344. ISSN 07366906. Dostupné z: http://portal.acm.org/citation.cfm?doid=333329.333344

[7] Virtual Reality in Healthcare. *Virtual Reality Society* [online]. 2017 [cit. 2017-04-03]. Dostupné z: https://www.vrs.org.uk/virtual-reality-healthcare/

[8] Virtual Reality in Military. *Virtual Reality Society* [online]. 2017 [cit. 2017-04-03]. Dostupné z: https://www.vrs.org.uk/virtual-reality-military/

[9] HAAR, René ter. *Virtual Reality in the Military: Present and Future.* 2005. Faculty of Electrical Engineering, Mathematics and Computer Science University of Twente, the Netherlands.

[10] Buy Lumion. *Lumion* [online]. 2017 [cit. 2017-04-03]. Dostupné z: https://lumion3d.com/buy.html

[11] What is Unreal Engine 4. *Unreal Engine* [online]. Epic Games, 2017 [cit. 2017-04-03]. Dostupné z: https://www.unrealengine.com/what-is-unreal-engine-4

[12] Managing Content. *Unreal Engine Docs* [online]. Epic Games, 2017 [cit. 2017-04-03]. Dostupné z: https://docs.unrealengine.com/latest/INT/Engine/Content/index.html

[13] Material Editor Reference. *Unreal Engine Docs* [online]. Epic Games, 2017 [cit. 2017-04-03]. Dostupné z: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/Editor/

[14] TAVAKKOLI, Alireza. *Game development and simulation with Unreal technology.* ISBN 978-1498706247.

[15] Light Types. *Unreal Engine Docs* [online]. Epic Games, 2017 [cit. 2017-04-03]. Dostupné z: `https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/LightTypes/`

[16] SATHEESH, P. V. *Unreal Engine 4 Game Development* Essentials. United Kingdom: Packt Publishing Limited, 2016. ISBN 978-1784391966.

[17] SCHMALSTIEG, D. a Tobias HOLLERER. *Augmented reality: principles and practice.* Addison-Wesley usability and HCI series. ISBN 978-0321883575.

[18] SILTANEN, Sanni. *Theory and applications of marker-based augmented reality.* 2012. ISBN 9789513874490.

[19] About the Traditional Template Square Marker. *ARToolKit* [online]. [cit. 2017-04-04]. Dostupné z: `https://artoolkit.org/documentation/doku.php?id=3_Marker_Training:marker_about`

[20] An Introduction To Simultaneous Location and Mapping. *Kudan* [online]. 2017 [cit. 2017-04-20]. Dostupné z: `https://www.kudan.eu/kudan-news/an-introduction-to-slam/`

[21] DAVISON, Andrew J. Real-time simultaneous localisation and mapping with a single camera. In: *Proceedings Ninth IEEE International Conference on Computer Vision* [online]. IEEE, 2003, 1403-1410 vol.2 [cit. 2017-04-20]. DOI: 10.1109/ICCV.2003.1238654. ISBN 0-7695-1950-4. Dostupné z: `http://ieeexplore.ieee.org/document/1238654/`

[22] WLOKA, Matthias M. a Brian G. ANDERSON. Resolving occlusion in augmented reality. In: *Proceedings of the 1995 symposium on Interactive 3D graphics - SI3D '95* [online]. New York, New York, USA: ACM Press, 1995, s. 5-12 [cit. 2017-04-14]. DOI: 10.1145/199404.199405. ISBN 0897917367. Dostupné z: `http://portal.acm.org/citation.cfm?doid=199404.199405`

[23] SHAH, Manisah Mohd, Haslina ARSHAD a Riza SULAIMAN. Occlusion in Augmented Reality. In: *2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012).* Korea, 2012, 372 - 378. ISBN 978-1-4673-1288-2.

[24] HAUCK, Joao Vitor de Sá, Matheus R. F. MENDONCA a Rodrigo Luis de Souza da SILVA. *Occlusion of Virtual Objects for Augmented Reality Systems using Kinect.*, 2014, [cit. 2017-04-15], 5., Dostupné z: `http://www.gcg.ufjf.br/pub/doc92.pdf`

[25] SCHMALSTIEG, Dieter, Anton FUHRMANN, Gerd HESINA, Zsolt SZALAVÁRI, L. Miguel ENCARNAÇÃO, Michael GERVAUTZ a Werner PURGATHOFER. The Studierstube

Augmented Reality Project. *Presence: Teleoperators and Virtual Environments* [online]. 2002, 11(1), 33-54 [cit. 2017-04-18]. DOI: 10.1162/105474602317343640. ISSN 1054-7460. Dostupné z: http://www.mitpressjournals.org/doi/10.1162/105474602317343640

[26] About Augment. *Augment* [online]. Augment, 2017 [cit. 2017-04-03]. Dostupné z: http://www.augment.com/about-us/

[27] Who we are. *AkzoNobel* [online]. AkzoNobel, 2017 [cit. 2017-04-03]. Dostupné z: https://www.akzonobel.com/about-us/who-we-are

[28] The Dulux Visualizer App. *Dulux* [online]. AkzoNobel, 2017 [cit. 2017-04-03]. Dostupné z: https://www.dulux.co.uk/en/articles/the-dulux-visualizer-app

[29] About Us. *Kudan* [online]. 2017 [cit. 2017-04-13]. Dostupné z: https://www.kudan.eu/about/

[30] AR Features. *Vuforia* [online]. 2017 [cit. 2017-04-13]. Dostupné z: https://www.vuforia.com/Features

[31] SILTANEN, Sanni, Mika HAKKARAINEN a Petri HONKAMAA. *Automatic Marker Field Calibration.* 2007. VTT Technical Research Centre of Finland.

A Příloha na CD/DVD

Přiložené CD obsahuje vypracovanou demonstrační projekt v Unity a samostatný instalační soubor. Přiložen je také návod pro instalaci, spuštění a ovládání dané aplikace. DVD dále obsahuje kalibrační program knihovny ARToolKit pro získání parametrů kamery.

Obsah CD/DVD

- Android standalone - instalační balíček s aplikací pro systém Android
- Kalibrace - adresář obsahující program pro kalibraci a potřebné knihovny. Dále adresář obsahuje PDF soubor s kalibrační šachovnicí
- Manuál - soubor s návodem pro spuštění aplikace, dále se zde nalézá adresář s obrazy markerů pro tisk
- Unity projekt - soubory projektu pro engine Unity
- Windows standalone - obsahuje instalační soubor výsledné aplikace pro Windows